Incorporatie van a-priorikennis in diepe neurale netwerken voor regelaars van robots met poten

Incorporating Prior Knowledge into Deep Neural Network Controllers of Legged Robots

Jonas Degrave

Promotoren: prof. dr. ir. F. wyffels, prof. dr. ir. J. Dambre
Proefschrift ingediend tot het behalen van de graad van
Doctor in de ingenieurswetenschappen: computerwetenschappen

UNIVERSITEIT
GENT

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. K. De Bosschere
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2017 - 2018

ISBN 978-94-6355-111-3
NUR 984
Wettelijk depot: D/2018/10.500/29

Examination Board

prof. dr. Filip De Turck, chair
Department of Electronics and Information Systems
Faculty of Engineering and Architecture
Ghent University

prof. dr. Francis wyffels, supervisor
Department of Electronics and Information Systems
Faculty of Engineering and Architecture
Ghent University

prof. dr. Joni Dambre, supervisor
Department of Electronics and Information Systems
Faculty of Engineering and Architecture
Ghent University

prof. dr. Tony Belpaeme
Department of Electronics and Information Systems
Faculty of Engineering and Architecture
Ghent University

prof. dr. Stijn Derammelaere
Department of Electromechanics
Faculty of Applied Engineering
University of Antwerp

dr. Martin Riedmiller
Deepmind
London
United Kingdom

prof. dr. Eric Laermans
Department of Information Technology
Faculty of Engineering and Architecture
Ghent University

prof. dr. Guillaume Crevecoeur
Department of Electrical Energy, Metals, Mechanical
Constructions and Systems
Faculty of Engineering and Architecture
Ghent University

Acknowledgements

In here you will find my PhD dissertation, a document which would never have been created without the untiring help from a group of amazing people.

First and foremost, I would like to thank Ira for her unrelenting support. This would not have been possible without her. I also want to thank the rest of my family for supporting. Mama, papa, Jolien, Faith, Kytama & Saymen, it has been really appreciated.

I want also to thank Francis, as he has been my guide over the course of my PhD, even while he was still a PhD student himself. If it would not have benefit you, I would never have written this archaic, obsolete piece of a groupthink conformity ritual. Thank you for never using power as a managing method, but giving guidance instead. Not to forget the other members of the lab too, for the engaging discussions throughout the whole traject. I want to thank Joni for providing insight into the larger scope surrounding the research questions in this book. I especially want to mention Sander as well for patiently explaining us the whole deep neural networks paradigm. I think many people in the lab would not have been where they are today if it were not for his excellent dissemination.

I also want to thank the people at Deepmind for giving me an opportunity to go even deeper in the world of deep learning and robotics, and not minding too much that it took a little longer to finish writing up this dissertation.

And finally, I want to thank a lot of my friends, who have been supportive of my decision to start and have been pushing me to finish what I started. A big thanks to Pieter I, Kevin, Pieter G, Jens, Ilse, Robin, Laurens, Jolien, Glenn, Arne, Sangeeta en Lieven.

Jonas

List of Abbreviations

BPTT	Back Propagation Through Time
CMA-ES	Covariant Matrix Adaptation Evolution Strategy
CPG	Central Pattern Generator
CPU	Central Processing Unit
CZT	Chirp-Z Transform
DFT	Discrete Fourier Transform
DoF	Degrees of Freedom
ELM	Extreme Learning Machines
ESN	Echo State Networks
FFT	Fast Fourier Transform
GA	Genetic Algorithm
GAN	Generative Adversarial Networks
GPU	Graphics Processing Unit
GSP	Gauss Seidel Projection
ICA	Independent Component Analysis
IFFT	Inverse Fast Fourier Transform
IMU	Inertial Measurement Unit
LCP	Linear Complementarity Problem
LR	Linear Regression
LSM	Liquid State Machine
LSTM	Long Short-Term Memory
MSE	Mean Squared Error
PCA	Principal Component Analysis
PSO	Particle Swarm Optimization

RC	Reservoir Computing
RLS	Recursive Least Squares
RNN	Recurrent Neural Network
SFA	Slow Feature Analysis

Summary

Incorporating Prior Knowledge into Deep Neural Network Controllers of Legged Robots

This dissertation explores multiple ways of adding prior knowledge to neural networks used as controllers in robotics. It can largely be split into two parts. The first part of the dissertation focuses on adding prior knowledge to the gait generation, to spend less time in the optimization process to find efficient solutions. The second part of the dissertation focuses on the use of morphological computation as prior knowledge in the generation of stable gaits for legged robots.

In the introduction, we discuss in depth what is meant by prior knowledge in this context. We show how the concept of prior evidence emerges naturally from the creation of a probabilistic framework for 'degree of belief'. We then discuss how this prior evidence can be used in robotics using a paradigm built on an alternative view on computation, called morphological computation. We argue how this approach makes a natural match for controlling compliant robots.

There are multiple ways to add prior knowledge to neural networks. As a first step, in chapter 2 we explored data augmentation as a way to teach neural networks how to be invariant to affine image transformations. Image augmentations are a known way to have a convolutional neural network learn this invariance in natural images. We improve on this idea by putting the affine transform as a differentiable layer into

the neural network, thereby allowing the neural network to encode this invariance explicitly, rather than to have to encode this implicitly in the values of its parameters. The network is then able to transform images as a special type of layer, next to convolutional or dense layers. We show that explicitly encoding this prior knowledge of affine invariance into the architecture outperforms the previous method of using image augmentations.

Next, in chapter 3 we move our focus to robotics and develop three different gaits for the quadrupedal compliant robot *Oncilla*: a sine-based approach, a biologically inspired half ellipse approach and a spline-based approach. After comparing these approaches, we find that the method based on biological gaits is the most efficient of the three, especially at higher speeds. After this, we move our attention to approaches for turning. We showe the importance of scapulae for turning in quadrupedal robots. We also show that to be able to optimize the gaits without relying on a model, a lot of prior knowledge is needed to keep the time required for gait optimization low.

Consequently, in chapter 4 we evaluate whether transfer learning known gaits to gaits for new situations improves the optimization process. We analyzed this by starting the optimization process for various setups with gait parameters which had already been optimized for flat terrain. We find that it indeed works in most cases, and at least did not hurt the optimization process. We uncover that in this case, the reduced amount of exploration of the parameter space required before the parameters converges to an optimal solution is the reason for a warm start helping the optimization process. The optimization algorithm can, therefore, find good solutions faster, and fine-tune the parameters longer for a better end performance.

After this, we move our focus to morphological computation. As a first aspect in chapter 5, we study morphological sensing, and more specifically, whether we can use general purpose sensors available on a small legged robot to classify the underground it is walking on. Since the dynamics of the robot change with the underground it walks across, it should be possible to infer this underground from the sensors monitoring the body of the robot. Since we do not require any specialized sensors for the detection of the underground, we can argue that we are using the body of the robot as a resource of computation for the classification. We can indeed classify the underground successfully in

most cases, both with supervised and unsupervised algorithms. In a second part of chapter 5, we delve into which properties of the models are important for the correct classification. We find indications in our data that both memory and non-linearities are important aspects of this classification process and that they reinforce each other, which provides a starting point for the research in the next chapter.

Since gaits of legged robots are typically on the eigenfrequencies of their morphology, the morphology can probably be used as a resource for computation to generate the control signals. This is a concept called morphological control. In chapter 6 we are indeed able to move part of the control onto the morphology and show that there is a trade-off between the memory aspects and the non-linear dynamics needed for it to perform well. It seems that the main parameter is the number of uncorrelated signals the linear regression receives. The more signals with information, the better the performance and the smaller the error between the found closed loop controller and the target open loop trajectory. Using this, we are able to have the *Oncilla* perform a stable gait without requiring any memory, using an ELM setup to generate the motor signals from the sensors. We show that a stable closed loop limit cycle can be obtained using supervised learning for only a few periods of its gait, slowly transferring control from the open to the closed loop.

Finally, in chapter 7 we stretch the idea of morphological computation, and treat the whole legged robot with its controller as a single system to be optimized. We optimize it using a deep neural network as controller of a system by backpropagation through physics. To do this, we developed a physics engine framework inside an automatic differentiation library. This allows us to backpropagate through the controller, physics and renderer. We are able to show remarkably short optimization processes despite only having quite complex sensory signals such as cameras as inputs, in setups which are only partially observable and underactuated.

We conclude that incorporating prior knowledge is beneficial when setting up machine learning models for controlling robots. We also conclude that we were able to show that both morphological sensing and morphological control can be valid strategies for developing controllers for legged robots.

Samenvatting

Incorporatie van a priori kennis in diepe neurale netwerken voor regelaars van robots met poten

Deze dissertatie verkent de verschillende manieren om prior kennis toe te voegen aan neurale netwerken gebruikt als regelaar van robots. Het is verdeeld in twee grote delen. In het eerste deel van de thesis, focussen we ons op het toevoegen van a priori kennis aan het genereren van wandelgangen, zodat we minder tijd nodig hebben in het optimalisatieproces om goede wandelgangen te vinden. In het tweede deel ligt de focus op het gebruik van morfologische berekenen als a priori kennis om stabiele wandelgangen voor robots met poten te genereren.

In de introductie bediscussiëren we wat er bedoeld wordt met a priori kennis in de context van de dissertatie. We tonen hoe het concept a priori kennis natuurlijk te voorschijn komt wanneer je een probabilistisch raamwerk opzet rond 'graad van geloof in iets'. We gaan dan dieper in op hoe die a priori kennis kan gebruikt worden in de robotica via een paradigma gebaseerd op een alternatieve kijk op berekenen, genaamd morfologisch berekenen. We argumenteren hoe die benadering van het probleem een natuurlijke match is voor het regelen van meegevende robots.

Er zijn verschillende manieren om a priori kennis toe te voegen aan neurale netwerken. Als een eerste stap verkenden we in hoofd-

stuk 2 het gebruik van data-augmentatie als een manier om neurale netwerken te leren om invariant te zijn voor affiene transformaties van afbeeldingen. Afbeeldingsaugmentaties zijn een bekende manier om convolutionele neurale netwerken die invariantie aan te leren. We verbeterden dit idee verder door de affiene transformatie als afleidbare laag in ons neuraal netwerk te steken. Hierdoor is het neurale netwerk in staat om expliciet die invariantie te encoden, in plaats van het impliciet te moeten encoden in zijn parameters. Het netwerk is dan zelf in staat om de afbeeldingen te transformeren met een speciaal nieuw type van laag, naast convolutionele, samenvoeg- of dense lagen. We toonden aan dat het expliciet inbouwen van de affiene transformatie beter werkt dan het gebruik van data-augmentatie.

In hoofdstuk 3 vervolgens verplaatsten we onze focus naar robotica en ontwikkelden we drie verschillende wandelgangen voor de vierpotige, meegevende robot *Oncilla*: een op sinus gebaseerde benadering, een biologisch geïnspireerde benadering gebaseerd op halve ellipsen en een benadering gebaseerd op splines. Na het vergelijken van verschillende benadering om de controlesignalen te parametriseren, toonden we aan dat de methode die gebaseerd was op wandelgangen gevonden in biologie het efficiëntste was van de drie, vooral op hogere snelheden. Hierna bestudeerden we verschillende benadering om te draaien in een wandelgang en toonden we het belang van scapulae aan om te draaien in vierpotige robots. We toonden ook aan dat om de wandelgangen te kunnen optimaliseren zonder gebruik te kunnen maken van een model, er veel a priori kennis nodig is om de optimalisatietijd kort te houden.

In hoofdstuk 4 evalueerde we of de transfer van gekende wandelgangen naar wandelgangen voor nieuwe situaties het optimalisatieproces verbetert. Dit deden we door het optimalisatieproces van verschillende opstellingen te starten met parameters die al geoptimaliseerd waren voor een vlak terrein. We vonden dat dat inderdaad werkte in de meeste gevallen, en dat het het optimalisatieproces nooit schaadde. We achterhaalden dat de reden waarom de warme start werkte, was dat het de nood aan exploratie in de parameterruimte verkleinde vooordat de parameters konden convergeren naar de optimale oplossing. Daardoor werden er sneller goede parameters gevonden, en konden die parameters langer fijn afgesteld worden zodat het eindresultaat beter was.

In het tweede deel van de thesis verplaatsten we onze focus naar morfologische berekeningen. In hoofdstuk 5 deden we in eerste instantie onderzoek naar morfologisch waarnemen. Specifiek onderzochten we of we de algemene sensoren op een kleine robot met poten konden gebruiken om de ondergrond waarover het liep te classificeren. Aangezien de dynamica van de robot verandert afhankelijk van de ondergrond waarover het loopt, zou het mogelijk moeten zijn om het type achtergrond af te leiden van de sensoren op het lichaam van de robot. Aangezien we geen gespecialiseerde sensoren gebruiken voor het detecteren van het type ondergrond, zouden we dus kunnen argumenteren dat de robot als het ware gebruikt wordt als een middel om een deel van de berekeningen voor de classificatie over te nemen. Deze methode om te classificeren werkte inderdaad en in het tweede deel van het hoofdstuk gingen we dieper in op de modellen die we gebruikten voor de correcte classificatie. We toonden aan dat geheugen en niet-lineariteiten belangrijke aspecten zijn in dit classificatieproces en dat ze elkaar versterken, wat het startpunt vormt voor het onderzoek in het volgende hoofdstuk.

Aangezien de wandelgang van een robot typische gebeurt op de eigenfrequentie van zijn morfologie, kan de morfologie waarschijnlijk gebruikt worden als een bron voor berekeningen om de controlesignalen voor de motors te genereren. Dit is een concept dat morfologisch regelen wordt genoemd. In hoofdstuk 6 bleken we inderdaad in staat om een deel van de regelkring over te brengen naar de morfologie van de robot. Hierna toonden we aan dat er een afweging is tussen geheugenaspecten enerzijds en niet-lineariteiten anderzijds. Het bleek dat aantal niet-gecorreleerde signalen dat de lineaire regressie krijgt de belangrijkste parameter vormt. Hoe meer signalen met zinvolle informatie, hoe beter de prestaties en hoe kleiner de fout tussen de gevonden gesloten regelkring en het doeltraject. Aan de hand van die afweging, konden we een stabiele wandelgang creëren zonder extra geheugen te moeten toevoegen, aan de hand van een simpele ELM-opstelling die motorsignalen kan genereren uit de sensorsignalen. We toonden aan dat een stabiele limietcykel kon gevonden worden door gesuperviseerd te trainen in de loop van een paar wandelstappen, als je de regelkring langzaam interpoleert van de open regelkring naar de gesloten regelkring.

In hoofdstuk 7 tenslotte strekten we dit idee van morfologische berekeningen verder uit en behandelen de hele robot met poten en

zijn regelkring als één systeem om te optimaliseren. We optimaliseerden een diep neuraal netwerk als regelkring via het terug propageren door de fysica van het systeem. Om dit te kunnen doen, ontwikkelden we een physics engine binnen een bibliotheek voor automatische differentiatie. Dit liet ons toe om de fout terug te propageren door zowel de regelkring, de fysica van het systeem als de renderer. We konden aantonen dat een opvallend kort optimalisatieproces mogelijk was, ondanks dat we alleen vrij ingewikkelde sensorsignalen zoals camera's gebruikten om opstelling die enkel partieel observeerbaar en ondergeactueerd waren te regelen.

We concluderen dat het incorporeren van a priori kennis vrijwel altijd gunstig is als je gebruik maakt van machinaal geleerde modellen voor het regelen van robots. We komen ook tot de conclusie dat we konden aantonen dat zowel morfologisch waarnemen als morfologisch regelen valabele strategieën zijn om regelkringen voor robots met poten te ontwerpen.

Prologue

Rocks found on the bottom of rivers have no sharp edges, are rather smooth and could be described as being aerodynamic. These are exactly the features of which we know they reduce erosion, and thus allow the rock to remain as it is and where it is. I would argue that rocks are therefore well adapted to their environment, when the goal of the adaptation is to 'remain'.

Of course, the reason why rocks are so well adapted is so trivial it almost needs no explaining as to why this is the case. Suppose there were river rocks which were not well adapted to being in the river. They would either be picked up by the river and moved to the sea, or they would undergo a lot of erosion by the river, until they would either break or become more smooth and well adapted. So of those not well adapted rocks, a fraction would become well adapted to staying in the river, while the others would cease to exist. And that is why we find mostly well adapted river rocks.

This mechanism is so simple, it is often overlooked. Yet, it is also the most basic of optimization algorithms at work. When selecting random solutions to a problem and removing the bad ones, you tend to be left with better-than-random solutions.

This idea can be found back in many forms. In the context of problem solving, it is known as the brute force method to find a solution. In the context of optimization, it is known as random selection optimization, which finds its use mostly as a baseline method. Informally, this baseline is occasionally referred to as the chimpanzee level, af-

ter the idea that a chimpanzee on a typewriter could reproduce this dissertation, if only given enough time.

It was in this universe of random selection optimization, that a more efficient algorithm was found. According to the RNA world hypothesis, this process of random selection is powerful enough for autocatalytic chemicals to appear. These molecules increase the chance of a copy of themselves being created. Especially RNA, a chain of simple amino acids, would be a prime candidate as it – under the right circumstances – could accelerate the creation of copies of itself. Since these molecules essentially copy themselves, they are tried more than would be expected in a random selection optimization process.

And thus a speed-up begun. Things were not evaluated at random anymore, but things which copy themselves were more often tried for their propensity to remain. This process is known as evolution, as coined by Charles Darwin. In the context of optimization, it is known as evolutionary computation. The foundation underlying this idea, is that something which is the copy or a close copy of something successful, is more likely to be successful at a task as well.

And thus, evolution took off on this planet. First, some RNA-strings managed to catalyze amino acids to ease their own creation. Some catalyzed proteins which increased the copy speed. They specialized and cooperated to form more and more complex systems whose power to remain in this universe was even stronger, whether as itself or in the form of a copy.

Over multiple generations, this evolutionary process went through multiple major transitions. These major transitions involved changes in the way information is stored and transmitted, and what it is exactly, that is trying to remain. Nonetheless, the method of optimization stays the same. RNA evolved to an information-encoding role as DNA and moved the catalyst process to the proteins it created, the so-called enzymes. Prokaryotes grouped together and cooperated inside eukaryotes upon the discovery of the cell wall. These eukaryotes discovered sexual reproduction as a mean to speed up the evolution process. Groups of these eukaryotes clumped together and formed the first multicellular organisms. These multicellular organisms then formed societies. In turn, memes evolved within these societies.

Not all steps are fully understood, and often the outcome remains that more tangible proof is required before jumping to conclusions. Yet, we have a scientifically sound sketch as to how we could have come from autocatalyst chemicals which in turn are simple enough to have been created by a random selection process. This is not only we as humans, but we as humanity. We form a society where information is procreating through memes, outside of the inheritance mechanism. Memes such as the information contained in this doctorate.

Make no mistake. There is no reason to assume that these memes did not go through a process which could be described as evolution. Before the contemporary combustion engine meme was born, it went through many generations of failing or less efficient designs. The plan for this engine is the remainder of the best ideas from its predecessors.

However, because of reasoning, many possible variations were never even taken into consideration in this whole process, without affecting its outcome. So, it can be noted that since we only need to try a millions of memes in the course of a couple of hundred of years to develop remarkably complex systems, we have sped up the evolution process tremendously. In an age where first time success of designs is slowly becoming the norm, evolution is reaching a threshold point. Because, what if we already know which step we should evaluate next as it will almost guaranteed be more optimal than what we have now? What if instead of sampling from variations of a concept, and seeing which one works best, we could directly derive the most optimal direction we should delve into?

Such optimizations are known as higher order iterative methods. Many algorithms are known within this category. They all rely on estimating information on the best direction for each parameter, without explicitly evaluating the directions for each parameter.

We, as science, a collection of memes within humanity, might be standing on the threshold to a new era of optimizing systems. For the first time, it seems possible that at some point in the near future, we might create systems which can improve themselves directly, without the need for trying multiple variations by human intervention. These systems would still step from generation to generation, but since they have no need to consider but a fraction of their variations, the speed of its optimization would outperform any imaginable form of evolu-

tion. This is especially true, since with an increasing complexity of the system, there is an exponentially increasing number of variations to consider.

I would argue that this event is as fundamental to the universe as life itself. It is a new threshold in the continuum of intelligence between river rocks and us as an intelligent rock floating through space. And it is within the context of this search for a general purpose system, that I hope my dissertation is putting a dent in the knowledge of humanity. Not with the goal to uplift myself as a person over my human condition, but to acknowledge the fundamental and universal desire also driving me. The desire to create things which remain.

Bibliography

[1] H. Douglas, *Gödel, Escher, Bach: An eternal golden braid.* Basic Books, 1979.

[2] R. Dawkins, *The selfish gene.* Oxford university press, 1976.

[3] J. M. Smith and E. Szathmáry, *The major transitions in evolution.* Oxford University Press, 1997.

Contents

List of Figures

List of Tables

1 Introduction

In the following sections, we will discuss modelling, as well as explain what is meant by prior knowledge in a machine learning context. This will then in turn allow us to lay out the research we did and explain the main foundations underlying our approach.

1.1 Machine Learning

1.1.1 On Modelling

Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed [1]. While it has been one of the more fruitful areas in recent years, it is still only able to tackle a specific subsection of the problems posed in the area. Contemporary machine learning algorithms are focused on learning from and making predictions on data. They do this not with strictly static program instructions, but by making data-driven predictions or decisions through building a model from sample inputs.

This approach is similar to the constructive approach used in science. In an idealized scientific method, in order to explain a system, we perform the following steps:

1. we formulate a theorem or model,
2. we observe the system we are analyzing and collect data,

3. we try to use the data to refute our model. If we refute the model, we go to the first step. If not, we go to the second step.

The more data is collected, the more certain we can be that our theorem does indeed describe the system we are analyzing. Note that this process never ends, we can never be certain that our theorem describes the system.

Note that in this case, a chronology is introduced where we first formulate a theorem, and only then collect data. This is done in order to find a good explanation of our system, rather than *an* explanation of the system. There are namely many possible explanations of data, but there are only a couple which will produce meaningful predictions or useful decisions.

In machine learning, typically, an alternative method is used:

1. we formulate a model, with a number of tunable parameters,
2. we observe the system we are analyzing and collect data,
3. we optimize the model parameters to our data,
4. we observe the system again and collect more data,
5. we try to use the data to refute our model. If we refute the model, we go to the first step. If not, we go to the third step.

In this approach, we use observations of our system to construct a better model. This is partly done manually (by formulating a model with parameters) and partly in an automatic fashion (by optimizing the parameters). Note the importance of keeping validation data separately to test the model on the data. This is important to make sure our model will be able to generate useful predictions.

The benefit of this approach is that a lot of manual labor is taken away, and replaced by an optimization process which can be automated. One should proceed carefully however, as a model with a sufficient amount of parameters can model any finite amount of data. These explanations by the model are not necessarily good and that is why the additional step to weed out the badly parameterized models is required.

1.1.2 Probability and Prior Knowledge

As illustrated in the previous section, models are built in a two step process. First, we formulate a model, and then we optimize its parameters. We initially pretended that the first step can be done out of the blue, but this is of course not true. When building initial models, we try to encode for what we know, just as is the case in the scientific method. Einstein did not formulate the formulations of his relativity theory out of the blue, he carefully crafted them from basic principles. The same is true for modelling using machine learning.

What actually happens, is that assumptions are made prior to the creation of a model. Since the assumptions do not come from observed data, they come from a source separate that observations of the system. All this information about a problem which is available outside of training data, is known as 'prior knowledge' [2].

This idea of prior information, is closely tied to the foundations of probability. For this reason, we include a small introduction to probability, and in the following sections, we will describe how this notion of prior knowledge arises. This is especially interesting, as modelling systems found in the real world, is almost inherently probabilistic.

1.1.2.1 On defining probability

There are many ways in which probabilities are introduced. Firstly, some events are inherently probabilistic, such as the throwing of dice. Secondly, observations of the world introduce a measurement error, such as the inaccuracies when measuring the height of someone. Thirdly, we assign probabilities to arbitrary future events, such as that it is more likely for the first brain transplant to happen in China.

Notice that these three views on probability are three semantically different things. The first could be described as the odds of something happening, the second as an error on a measurement, and the third as the certainty of a statement. Yet, all three of them are tied to a probabilistic interpretation. This can be confusing, and it therefore deserves to put some focus on what we mean exactly by the word

'probability'.

There is the formal definition for the mathematical concept of probability. It defines a probability as the measure of a certain set, compared to the same measure on a super set containing the first set. It was axiomatized by Kolmogorov [3]. Let Ω be a non-empty set. A field on Ω is a set F of the subsets of Ω that has Ω as a member and that is closed under union and complementation with respect to Ω. Let P be a function from $F \to \mathbb{R}$ with the following properties:

1. $P(A) \geq 0, \forall A \in F$.
2. $P(\Omega) = 1$.
3. $P(A \cup B) = P(A) + P(B)$, $\forall A, B \in F$ where $A \cap B = \varnothing$.

Then, P is a probability function, and (Ω, F, P) is a probability space. These definitions can also be extended to cover sets of infinite size.

It is currently orthodoxy that probability on finite sets is defined like this, and that it also describes probability in the real world. Porting this definition to the world is however problematic from a philosophical point of view. It is hard to define what the superset of all possible events would be, and whether such a constructed notion could even be considered real. After all, not all of the events will actually happen. Therefore, there needs to be basis on how we can tie these probabilities to the physical world.

Some of these different views on probability are laid out in the following subsections. Note that although they all end up in the same mathematical framework, they have different ways of viewing how that framework relates to the real world. The following is also not meant as a fully elaborated view on all perspectives on probability, but rather a short introduction to some of the most influential views.

1.1.2.2 Classical Probability

Classical Probability goes back on ideas from Laplace, Pascal, Bernoulli, Huygens, and Leibniz on how to deal with games of chance. It assigns probabilities in the absence of any evidence, or in the presence of symmetrically balanced evidence. According to this view on probability, probability is shared equally among all possible outcomes,

such that the classical probability of an event is the fraction of the number of possibilities in which the event occurs, to the number of possibilities [4].

This view on probability is problematic, in that it is hard to find all possible outcomes such that it is indeed correct to assume equal probability. For example, we cannot define the probability of having rain tomorrow this way. Because the only two clear states for this are either "raining" and "not raining", both of which cannot be attributed probabilities, which does not correspond to our intuition on probability.

So this idea on probability has the inherent issue of not allowing to find out what the probability of events are. On the other hand, it does provide a framework to reason with probabilities once probabilities have been appointed to events, for instance when events can be considered equally probable through symmetry.

1.1.2.3 Objectivists frequentist probability

To alleviate these problems, frequentist probability has been developed by Venn [5] based on work from people such as Poisson, Ellis, Cournot and Fries. It identifies probability as the limit of the event happening over the number of trials, when the number of trials tends to infinity. Therefore, when n_A is the number of trials where the event A is recorded in a total number of trials N, then the probability $P(A)$ of an event A occuring is exactly

$$P(A) = \lim_{N \to \infty} \frac{n_A}{N}. \tag{1.1}$$

This is an operational definition, which works for large parts of the sciences. It allows us to reason on probability, and even measure probability. This view on probability is strongly tied to the physical world, and is therefore called 'objectivist'.

There are however some limitations to this idea, not the least of which is that we can only measure probability in repeatable processes. We can indeed measure the probability of rain tomorrow, but we cannot talk about the probability of the first human on Mars being female.

The latter event can only happen once. Therefore, before it actually happens, the frequentist approach offers no framework to reason on its probability.

1.1.2.4 Subjectivist epistemic probability

A solution to this came from the Dutch book thought experiment by de Finetti [6]. In this thought experiment, you must set the price p for having to pay 1 if event A happens. Another person then gets the possibility to choose which side of the bet to take, the side of the bookmaker or the side of the bettor. So either you pay p now and he might pay you 1 if A happens, or he pays you p now and you might have to pay him 1 if A happens. You set the prices p for all events A in a predetermined set Ω.

De Finetti showed that if your bets are not coherent, i.e. they do not follow Kolmogorov's axioms, that the second person could always make a so-called 'Dutch book'. This is a set of bets which guarantees him to win, irrespective of the results of the events in Ω. If such a book cannot be made, then the probabilities p appointed to the events in Ω are coherent. Therefore:

$$P(A) = p. \tag{1.2}$$

This view offers an intuitive way of tying probabilities to events. Probabilities here can be seen as 'confidence' or a 'degree of belief'. This is different from the objectivist 'rate of observation' or the classical 'possibility' [7].

From this Dutch book thought experiment, one could start reasoning on how observations can affect the agents betting, and thus the probabilities. What happens to the probability when both actors are making bets while observing evidence E?

In such case, one must make sure that there is no diachronic book available. That is, there exists no method which guarantees that at the end of all bets made a Dutch book is formed, whether the bets are made before or after observing E. This implies that the probabilities must evolve over time, based on the evidence E available.

Since in this sense, the evidence E is having an effect on the probability, we write this Bayesian probability after observing E as

$$P(A|E) = p. \tag{1.3}$$

The evidence E is referred to as the 'prior'. It encapsulates all knowledge available to the agent making the bets. This knowledge can – and generally does – influence the subjective probability an agent would assign to events.

One of the goals of this dissertation, is to assess how we can use our prior knowledge on the problem in order to create better models. We want to find ways such that we can correctly adapt the probabilities in the predictions of our models to the evidence which is available to us. Or less abstract, we wish that our model has a more accurate 'degree of belief' with regards to the evidence we have available.

Note that we never had to be explicit on the nature of this evidence E. Some of this evidence E expresses specific, definite information about a certain variable in the problem at hand, such as 'the mean is 0'. It could however only express vague or general information as well, such as 'your observations will be positive'. The former is called an informative prior, the latter an uninformative or diffuse prior. We will mainly focus on on transferring uninformative priors into our models. In chapter 2 we will discuss a method we developed to add more image priors into neural networks for image classification. In chapter 4 we will discuss transfer learning, a method where prior knowledge obtained from optimizing a different problem is used in the optimization process of another algorithm.

In the following section, we will shift focus and start introducing robotics. However, within robotics we will go looking for the parts of prior knowledge we could use when developing neural networks as controllers, e.g. using embodiment to exploit the dynamics already available in the robot for the generation of stable gaits.

1.2 Robotics

A robot is a machine capable of carrying out a diverse range of complex tasks through series of actions automatically. The underlying goal of research in robotics, is therefore to (partially) take over tasks generally done by humans and make their work lighter or irrelevant.

In this sense, contemporary robots are often found in the context of manufacturers where they replace tasks previously done by human labor. Initially, the development of robotics has focused on tasks which are harming the human in the process, because the task is to arduous, dangerous or monotonous. Think respectively of welding robots in the automotive industry, robots for nuclear inspection or pick-and-place robots.

Over time, robots were able to achieve superhuman performance regarding precision, power, reliability and cost for various tasks. In the automotive industry, it is not uncommon to have robots manipulate and weld objects with masses of hundreds of kilograms to millimeter accuracy. They do this for weeks on end, requiring little outside intervention.

However, robots still underperform when it comes to interacting or cooperating with humans, perception tasks and flexibility of operation. Indeed, most robots are still found inside cages to separate them from human co-workers. On top of that, robots still have difficulty perceiving their environment, and thus have difficulty dealing with unforeseen circumstances in this environment. Lastly, whereas we already mentioned that robots can be cost-effective, this is not necessarily the case when you need control engineers to optimize an entire assembly for a new product. In a flexible manufacturing line, humans are economically still more practical and cost-effective.

The reasons behind this underperformance are harder to point out. For one, actuators are still severely lacking compared to those of humans. The actuators used tend to be either power-hungry, heavy or unreliable. Right now, the standard actuators found are either hydraulic or electric, and both tend to have a high inertia.

But this is only part of the issue. A second related issue is that robots also have difficulty dealing with compliance and compliant objects.

This is a result of the fact that deformable objects are hard to model. Yet, deformable and soft objects are found all over in our environment. Humans themselves tend to be quite deformable. Moreover, a part of the safety problem with robots, is due to them being very rigid. So, how robots can function despite working in a non-rigid, compliant world is a problem which needs to be tackled.

A third problem, is that robots still have difficulties in sensing their environment effectively. Sensors are often big and cumbersome, and need cabling and power to process the incoming information.

A fourth problem, is that reliability is usually coming with weight. In the case of moving parts of the robot, this means more power is needed to move this part. On top, extra weight means extra inertia as well. In the case of legged robots, where every additional gram of weight makes the task more difficult, this means that a balance must be found between weight and reliability.

There are of course more problems than the four described above, such as the difficulty for social interaction with humans [8] or the limited ability to self-repair [9]. But the four problems above are the focus in this dissertation.

1.2.1 Compliant Robots

Since the 2000s, lot of recent research has gone into compliant robots in order to deal with the issues described before [10]. Compliant robots are a type of robot, where the rigidness typically associated with robotics is avoided. This can be done in order to increase the safety, power efficiency, robustness or simplicity of the control.

In this field, a distinction is usually made between two types of compliance:

- **Active compliance** is a subclass of compliance in which sensor input is used to control the actuators in a control loop to behave compliant. The downside of this approach, is that disturbances at a frequency higher than the control frequency cannot be covered.

- **Passive compliance** is a subclass of compliance, where springs and dampers are used to add compliance in the morphology of the robot. In this approach, high frequency disturbances can also be dealt with in a compliant way. The downside is that it is usually harder to remove this high frequency compliance when that would be required, making the robots react less snappily.

In recent years, compliant robots are increasingly gaining interest in the scientific community. They use low-impedance mechanisms to exploit the robot's passive dynamics and nudge it into the desired behavior, rather than enforcing a desired trajectory [11]. This approach is promising, as it is more closely aligned with biology, where even today animals show capabilities unmatched in robotics. Like in nature, we can make our robots more compliant by adding passive elements in the design.

These passive elements can for instance be used for storing energy between steps, such as with the walking Lucy robot [12], or the added compliance can be used for having more robust control of the robot on difficult terrain [13]. They can make control easier, such as the salamander robot where gaits for navigating land and sea were emerging from the same control approach [14]. They can make robots safer for use next to humans, such as was demonstrated by Festo's bionic handling assistant [15].

Examples of compliant robots include robots with compliant actuators, such as the CoMaN [16] and the Kuka-DLR light weight arm [17]. Notable examples of compliant robots in legged robotics are for instance the M2V2 [18], HyQ [19] and StarlETH [20].

More recently, the approach has been evolving towards 'soft robotics', where hard materials are shunned altogether [21]. Examples include the various octopus robots [22, 23], fish [24] and (quadrupedal) starfish [25]. Because of their intricate design, these robots are able to navigate very narrow spaces.

Despite their advantages, compliant robots are harder to control using linear control methods. Since their elastic elements show a non-linear behavior, control by classical paradigms requiring linearization of the problem prove to be less effective [26]. One approach to tackle this problem, is to have the robot learn its behavior or model using opti-

mization techniques from machine learning [27, 28, 29, 30, 31, 32, 33]. This way the robot inherently learns to deal with non-linearities and uncertainties in its own morphology. These uncertainties are an inherent problem in robotics, where the quality of contemporary sensors makes it impossible to know either the body of the robot or the environment in full detail.

Having said that, the main disadvantage with these techniques is that the robot needs to go through the optimization process for its behavior repeatedly, every time when the setting of the problem changes. This process takes up a lot of optimization time, and the time of testing a behavior on the robot is usually far larger than the amount of time spent on the calculations of the optimization process. It would therefore be opportune to make these optimizations more data-effective, reducing the number of behaviors to test and thus significantly reducing the amount of time needed to optimize. **In this thesis, we will demonstrate a number of ways in which prior knowledge of the problem at hand can be introduced in the optimization process, to make the optimization process more effective.**

1.2.2 Embodiment and Morphological Computation

The main approach we will focus on, is based on the idea of embodiment. This not in the trivial sense, where "intelligence requires a body", but in that a physical agent affects the environment and can in turn be influenced again by it through sensing [34]. In animals in the world, where the control and the physiology has evolved together, this tight coupling between sensors, the neural system, actuators and limbs is omnipresent. This in contrast to the common practice in robotics to separate the physical robot from its control, a separation often explicit as the controller is not integrated in the robot hardware but put on a separate box.

In this embodiment, we will focus on morphological computation. Morphological computation is the practice where certain parts of the computation generally left over to the controller, have been taken over by the materials, shape or the sensor morphology in the robot [34]. Compared to classical robotics, this is a radical change of perspective, as the complex dynamics which are usually surpressed, are exploited

in this paradigm.

In short, the term *morphological computation* is coined through the idea that in order for a robot such as Asimo to take a step, it requires intensive calculations in order to set out a trajectory where it can remain stable. On the other hand, in more compliant robots stability is almost guaranteed through the way the body has been designed. Now there are two equivalent ways to look at how the problem has been handled [35].

- Either the use of robot morphology has lead to a reduced requirement for the total amount of computation required. The robot has increased its controllability.
- Or in an equivalent view, the total amount of computation has remained constant, but the body has taken over some part of the computation originally performed by the brain. This part of the information processing is now called 'morphological computation'.

Of course, this computation has little involvement with computation as typically defined through the Church-Turing thesis, where inputs and outputs are assumed to be digitally encoded. Therefore an alternative model coined "natural computing" was coined to describe this alternative formalism for computation. In this setting, computation is seen as being performed by dynamical systems processing streams of input signals into output signals [36, 37].

One needs to be careful here not to descend into pancomputationalism [35]. In this view, as the universe is essentially a dynamical system, everything in it is inherently computing. At that point, the term 'computation' loses relevance and is merely a metaphor for a process. A critical point is therefore the encoding and decoding of the representation encoded in the dynamical system, defined before the dynamical system is set going. Therefore, some forms of morphological computation might more accurately be coined morphological control or morphological sensing.

The research question of this dissertation, is to assess whether and how prior knowledge available in the morphology of a robot can be put to use in common robotics tasks

such as sensing the environment or generating motor signals. We will discuss both morphological control and morphological sensing in chapters 5 and 6 respectively. We will stretch this idea to the extremum and treat the full physical dynamics with its controller as a single system in chapter 7. We treat this system as a single optimizable equation which we optimize in simulation with backpropagation of the complete system.

1.3 Scientific Contributions

This section summarizes the research challenges and scientific contributions made in the following chapters of this dissertation.

Chapter 2: Spatial Chirp-Z Transformer Networks

Convolutional Neural Networks are often used for computer vision solutions, because of their inherent modeling of the translation invariance in images. In this chapter, we propose a new module to model rotation and scaling invariances in images. To do this, we rely on the chirp-Z transform to perform the desired translation, rotation and scaling in the frequency domain.

This approach has the benefit that it scales well and that it is differentiable because of the computationally cheap sinc-interpolation. We show that the prior knowledge on image invariances by encoding it in the network architecture allows to increase performance in neural network models. This paper is a primer on adding differentiable layers into neural networks for encoding prior knowledge on the data, a concept which is further expanded upon in chapter 7. This chapter also shows the improvement in performance possible by including prior information when training neural networks.

Chapter 3: Comparing Trotting and Turning Strategies on the Quadrupedal Oncilla Robot

In this chapter, we compare three different trotting techniques and five different turning strategies on a small, compliant, biologically inspired quadrupedal robot: the Oncilla. The locomotion techniques were optimized on the actual hardware using a treadmill setup, without relying on a model of the robot.

We found that using half ellipses as foot trajectories resulted in the fastest gaits, as well as the highest robustness against parameter changes. Furthermore, we analyzed the importance of using the scapulae for turning, from which we observed that although not necessary, they are needed for turning with a higher speed. These gaits formed the basis for further research in chapters 4 and 6.

Chapter 4: Transfer Learning of Gaits on a Quadrupedal Robot

Learning new gaits for compliant robots is a challenging multi-dimensional optimization task. Furthermore, to ensure optimal performance, the optimization process must be repeated for every variation in the environment, e.g. for every change in inclination of the terrain. This is unfortunately not possible using current approaches, since the time required for the optimization is simply too high. Hence, a sub-optimal gait is often used.

The goal in this chapter is to reduce the learning time of a particle swarm algorithm, such that the robot's gaits can be optimized over a wide variety of terrains. To facilitate this, we use transfer learning by sharing knowledge about gaits between the different environments. This way, prior knowledge discovered during a previous optimization process can be reused in a new optimization process.

Our findings indicate that using transfer learning, new robust gaits can be discovered faster compared to traditional methods which learn a gait for each environment independently. Since we also find that this approach sometimes decreases efficiency, it is clear that using prior knowledge using is not always beneficial. It might harm the

optimization process when the new task is not sufficiently close to the previous task, such that the prior transferred is inaccurate.

Chapter 5: Terrain Classification for a Quadruped Robot

Using data retrieved from the Puppy II robot at the University of Zurich (UZH), we show that machine learning techniques with non-linearities and fading memory are effective for terrain classification, both supervised and unsupervised, even with a limited selection of input sensors.

We find that the classification error is small enough to have a robot adapt the gait to the terrain and hence move more robustly. The results indicate that most information for terrain classification is found in the combination of tactile sensors and proprioceptive joint angle sensors. Secondly, the results indicate the possible power of embodiment and morphological computation. Despite not having a dedicated sensor for classifying the terrain, we found that the difference in behavior of the limbs computed the features needed to classify the terrain already. Thirdly, the results indicate the trade-off between non-linearities and fading memory, a trade-off further explored in chapter 6.

Chapter 6: Developing an Embodied Gait on a Compliant Quadrupedal Robot

Incorporating the body dynamics of compliant robots into their controller architectures can drastically reduce the complexity of locomotion control by exploiting the morphological computation going on in the body of the robot. An extreme version of this embodied control principle was demonstrated in highly compliant tensegrity robots, for which stable gait generation was achieved by using only optimized linear feedback from the robot's sensors to its actuators. The morphology of quadrupedal robots has previously been used in chapter 5 for sensing and for control of a compliant spine, but not for gait generation.

In this chapter, we apply embodied control to the compliant,

quadrupedal Oncilla robot. As initial experiments indicated that mere linear feedback does not suffice, we explore the minimal requirements for robust gait generation in terms of memory and nonlinear complexity. Our results show that a memory-less feedback controller can generate a stable trot by learning the desired nonlinear relation between the input and the output signals. We believe this method can provide a robust tool for transferring knowledge from open loop to closed loop control on compliant robots.

Chapter 7: A Differentiable Physics Engine for Deep Learning in Robotics

In this chapter, we expand on the ideas for embodied control and we take a look at the culmination of combining robot morphology and controller into one system to optimize. Currently, robots are often treated as a black box in this optimization process, which is the reason why derivative-free optimization methods such as evolutionary algorithms or reinforcement learning are omnipresent. When gradient-based methods are used, models are kept small or rely on finite difference approximations for the Jacobian. This method quickly grows expensive with increasing numbers of parameters, such as found in deep learning. We propose an implementation of a modern rigid body physics engine, which can differentiate control parameters similar to the concept described in chapter 2. This engine is implemented for both CPU and GPU. Using this engine, we can backpropagate through time in our controller, through the physics of our engine and also through the rendering process.

This chapter shows how such an engine speeds up the optimization process, even for small problems. We argue that this is a big step for deep learning in robotics, as it opens up new possibilities to optimize robots, both in hardware and software. By backpropagating the error through a model of the robot, we are essentially transferring prior knowledge on the model of the robot into the neural network controller. We are able to show that it only takes a small number of update steps before the robot can learn to control a simple setup through vision, despite that the controller has around one million parameters.

1.3.1 Publications

1.3.1.1 Journal Publications

Jonas. **Degrave**, M. Burm, P.-J. Kindermans, J. Dambre, and F. wyffels, "Transfer learning of gaits on a quadrupedal robot," *Adaptive Behavior*, vol. 23, no. 2, pp. 69–82, 2015

G. Urbain, **Jonas**. **Degrave**, B. Carette, J. Dambre, and F. wyffels, "Morphological properties of mass–spring networks for optimal locomotion learning," *Frontiers in neurorobotics*, vol. 11, 2017

T. Vets, **Jonas**. **Degrave**, L. Nijs, F. Bressan, and M. Leman, "Plxtrm: Prediction-led extended-guitar tool for real-time music applications and live performance," *Journal of New Music Research*, vol. 46, no. 2, pp. 187–200, 2017

I. Korshunova, P.-J. Kindermans, **Jonas**. **Degrave**, T. Verhoeven, B. H. Brinkmann, and J. Dambre, "Towards improved design and evaluation of epileptic seizure predictors," *IEEE Transactions on Biomedical Engineering*, 2017

1.3.1.2 Conference Papers

Jonas. **Degrave**, R. Van Cauwenbergh, F. wyffels, T. Waegeman, and B. Schrauwen, "Terrain classification for a quadruped robot," in *12th International Conference on Machine Learning and Applications (ICMLA), 2013*, vol. 1. IEEE, 2013, pp. 185–190

Jonas. **Degrave**, M. Burm, T. Waegeman, F. wyffels, and B. Schrauwen, "Comparing trotting and turning strategies on the quadrupedal oncilla robot," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2013, pp. 228–233

Jonas. **Degrave**, K. Caluwaerts, J. Dambre, and F. wyffels, "Developing an embodied gait on a compliant quadrupedal robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015*. IEEE, 2015, pp. 4486–4491

Jonas. **Degrave**, S. Dieleman, J. Dambre, and F. wyffels, "Spatial

chirp-Z transformer networks," in *European Symposium on Artificial Neural Networks (ESANN)*, 2016

1.3.1.3 Conference Abstracts

Jonas. **Degrave**, F. wyffels, T. Waegeman, P.-J. Kindermans, and B. Schrauwen, "Applying morphological changes during the evolution of quadruped robots results in robust gaits," in *21st Annual Belgian-Dutch conference on Machine Learning (BeNeLearn & PMLS), 2012*. University Press, 2012, pp. 63–63

A. Sproewitz, A. Tuleu, M. D'Haene, R. Möckel, **Jonas**. **Degrave**, M. Vespignani, S. Gay, M. Ajallooeian, B. Schrauwen, and A. J. Ijspeert, "Towards dynamically running quadruped robots: performance, scaling, and comparison," in *Adaptive Motion of Animals and Machines*, 2013, pp. 133–135

Jonas. **Degrave**, J. Burms, I. Korshunova, and J. Dambre, "Using deep learning to estimate systolic and diastolic volumes from MRI-images," in *Benelearn*, 2016

B. Willems, **Jonas**. **Degrave**, J. Dambre, and F. wyffels, "Quadruped robots benefit from compliant leg designs," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017)*. IEEE, 2017

1.3.1.4 White Papers and Blog Posts

S. Dieleman, A. van den Oord, I. Korshunova, J. Burms, **Jonas**. **Degrave**, L. Pigou, and P. Buteneers, "Classifying plankton with deep neural networks," mar 2015

S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, D. M. de Almeida, B. McFee, H. Weideman, G. Takács, P. de Rivaz, J. Crall, G. Sanders, K. Rasul, C. Liu, G. French, and **Jonas**. **Degrave**, "Lasagne: First release," aug 2015

Jonas. **Degrave**, M. Hermans, J. Dambre, and F. wyffels, "A differ-

entiable physics engine for deep learning in robotics," *arXiv preprint arXiv:1611.01652*, 2016

F. Godin, **Jonas**. **Degrave**, J. Dambre, and W. De Neve, "Drelus: Dual rectified linear units," *arXiv preprint arXiv:1707.08214*, 2017

Bibliography

[1] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[2] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[3] A. N. Kolmogorov, "Grundbegriffe der wahrscheinlichkeitsrechnung, berlin, 1933," *English translation, Chelsea, New York*, 1950.

[4] A. Hájek, "Interpretations of probability," in *The Stanford Encyclopedia of Philosophy*, winter 2012 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2012.

[5] J. Venn, *The logic of chance: an essay on the foundations and province of the theory of probability*. Macmillan and Company, 1876.

[6] B. d. F. Roma, "Sul significato soggettivo della probabilità." ≪*Fundamenta Mathematicae*, vol. 17, pp. 298–329, 1931.

[7] W. Talbott, "Bayesian epistemology," in *The Stanford Encyclopedia of Philosophy*, winter 2016 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2016.

[8] I. Leite, C. Martinho, and A. Paiva, "Social robots for long-term interaction: a survey," *International Journal of Social Robotics*, vol. 5, no. 2, pp. 291–308, 2013.

[9] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable

robot systems [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[10] R. Pfeifer, M. Lungarella, and F. Iida, "The challenges ahead for bio-inspired'soft'robotics," *Communications of the ACM*, vol. 55, no. 11, pp. 76–87, 2012.

[11] G. A. Pratt, "Legged robots at MIT: what's new since raibert?" *Robotics & Automation Magazine, IEEE*, vol. 7, no. 3, pp. 15–19, 2000.

[12] B. Verrelst, R. Van Ham, B. Vanderborght, F. Daerden, D. Lefeber, and J. Vermeulen, "The pneumatic biped "lucy" actuated with pleated pneumatic artificial muscles," *Autonomous Robots*, vol. 18, no. 2, pp. 201–213, 2005.

[13] M. Raibert, K. Blankespoor, G. Nelson, and R. Playter, "Bigdog, the rough-terrain quadruped robot," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 10 822–10 825, 2008.

[14] A. J. Ijspeert, A. Crespi, D. Ryczko, and J.-M. Cabelguen, "From swimming to walking with a salamander robot driven by a spinal cord model," *science*, vol. 315, no. 5817, pp. 1416–1420, 2007.

[15] A. Grzesiak, R. Becker, and A. Verl, "The bionic handling assistant: a success story of additive manufacturing," *Assembly Automation*, vol. 31, no. 4, pp. 329–333, 2011.

[16] N. G. Tsagarakis, Z. Li, J. Saglia, and D. G. Caldwell, "The design of the lower body of the compliant humanoid robot "cCub"," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 2035–2040.

[17] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schäffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald, and G. Hirzinger, "The KUKA-DLR lightweight robot arm-a new reference platform for robotics research and manufacturing," in *41st international symposium on Robotics (ISR) and 6th German conference on robotics (ROBOTIK)*. VDE, 2010, pp. 1–8.

[18] J. Pratt and B. Krupp, "Design of a bipedal walking robot," in *SPIE Defense and Security Symposium*. International Society for Optics and Photonics, 2008, pp. 69 621F–69 621F.

[19] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D. G. Caldwell, "Design of hyq–a hydraulically and electrically actuated quadruped robot," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 225, no. 6, pp. 831–849, 2011.

[20] M. Hutter, C. Gehring, M. Bloesch, M. Hoepflinger, C. D. Remy, and R. Siegwart, "Starleth: A compliant quadrupedal robot for fast, efficient, and versatile locomotion," in *Int. Conf. on Climbing and Walking Robots (CLAWAR)*, 2012.

[21] D. Rus and M. T. Tolley, "Design, fabrication and control of soft robots," *Nature*, vol. 521, no. 7553, pp. 467–475, 2015.

[22] M. Wehner, R. L. Truby, D. J. Fitzgerald, B. Mosadegh, G. M. Whitesides, J. A. Lewis, and R. J. Wood, "An integrated design and fabrication strategy for entirely soft, autonomous robots," *Nature*, vol. 536, no. 7617, pp. 451–455, 2016.

[23] M. Cianchetti, M. Calisti, L. Margheri, M. Kuba, and C. Laschi, "Bioinspired locomotion and grasping in water: the soft eight-arm octopus robot," *Bioinspiration & biomimetics*, vol. 10, no. 3, p. 035003, 2015.

[24] A. D. Marchese, C. D. Onal, and D. Rus, "Autonomous soft robotic fish capable of escape maneuvers using fluidic elastomer actuators," *Soft Robotics*, vol. 1, no. 1, pp. 75–87, 2014.

[25] R. F. Shepherd, F. Ilievski, W. Choi, S. A. Morin, A. A. Stokes, A. D. Mazzeo, X. Chen, M. Wang, and G. M. Whitesides, "Multigait soft robot," *Proceedings of the National Academy of Sciences*, vol. 108, no. 51, pp. 20 400–20 403, 2011.

[26] B. K. Horn and M. H. Raibert, "Configuration space control," 1977.

[27] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun, "Learning motion patterns of people for compliant robot motion," *The International Journal of Robotics Research (IJRR)*, vol. 24, no. 1, pp. 31–48, 2005.

[28] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of*

the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 465–472.

[29] R. Dillmann, O. Rogalla, M. Ehrenmann, R. Zollner, and M. Bordegoni, "Learning robot behaviour and skills based on human demonstration and advice: the machine learning paradigm," in *International Symposium on Robotics Research (ISRR)*, vol. 9, 2000, pp. 229–238.

[30] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3. IEEE, 2004, pp. 2619–2624.

[31] J. R. Peters, "Machine learning of motor skills for robotics," Ph.D. dissertation, University of Southern California, 2007.

[32] T. Waegeman, F. wyffels, and B. Schrauwen, "Feedback control by online learning an inverse model," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 10, pp. 1637–1648, 2012.

[33] E. T. Wolbrecht, V. Chan, D. J. Reinkensmeyer, and J. E. Bobrow, "Optimizing compliant, model-based robotic assistance to promote neurorehabilitation," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 16, no. 3, pp. 286–297, 2008.

[34] R. Pfeifer and G. Gómez, "Morphological computation–connecting brain, body, and environment," *Creating brain-like intelligence*, pp. 66–83, 2009.

[35] V. C. Müller and M. Hoffmann, "What is morphological computation? on how the body contributes to cognition and control," *Artificial Life*, 2017.

[36] H. Hauser, A. J. Ijspeert, R. M. Füchslin, R. Pfeifer, and W. Maass, "Towards a theoretical foundation for morphological computation with compliant bodies," *Biological cybernetics*, vol. 105, no. 5, pp. 355–370, 2011.

[37] R. M. Füchslin, A. Dzyakanchuk, D. Flumini, H. Hauser, K. J. Hunt, R. H. Luchsinger, B. Reller, S. Scheidegger, and R. Walker, "Morphological computation and morphological control: steps

toward a formal theory and applications," *Artificial Life*, vol. 19, no. 1, pp. 9–34, 2013.

[38] Jonas. Degrave, M. Burm, P.-J. Kindermans, J. Dambre, and F. wyffels, "Transfer learning of gaits on a quadrupedal robot," *Adaptive Behavior*, vol. 23, no. 2, pp. 69–82, 2015.

[39] G. Urbain, Jonas. Degrave, B. Carette, J. Dambre, and F. wyffels, "Morphological properties of mass–spring networks for optimal locomotion learning," *Frontiers in neurorobotics*, vol. 11, 2017.

[40] T. Vets, Jonas. Degrave, L. Nijs, F. Bressan, and M. Leman, "Plxtrm: Prediction-led extended-guitar tool for real-time music applications and live performance," *Journal of New Music Research*, vol. 46, no. 2, pp. 187–200, 2017.

[41] I. Korshunova, P.-J. Kindermans, Jonas. Degrave, T. Verhoeven, B. H. Brinkmann, and J. Dambre, "Towards improved design and evaluation of epileptic seizure predictors," *IEEE Transactions on Biomedical Engineering*, 2017.

[42] Jonas. Degrave, R. Van Cauwenbergh, F. wyffels, T. Waegeman, and B. Schrauwen, "Terrain classification for a quadruped robot," in *12th International Conference on Machine Learning and Applications (ICMLA), 2013*, vol. 1. IEEE, 2013, pp. 185–190.

[43] Jonas. Degrave, M. Burm, T. Waegeman, F. wyffels, and B. Schrauwen, "Comparing trotting and turning strategies on the quadrupedal oncilla robot," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2013, pp. 228–233.

[44] Jonas. Degrave, K. Caluwaerts, J. Dambre, and F. wyffels, "Developing an embodied gait on a compliant quadrupedal robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015*. IEEE, 2015, pp. 4486–4491.

[45] Jonas. Degrave, S. Dieleman, J. Dambre, and F. wyffels, "Spatial chirp-Z transformer networks," in *European Symposium on Artificial Neural Networks (ESANN)*, 2016.

[46] Jonas. Degrave, F. wyffels, T. Waegeman, P.-J. Kindermans, and B. Schrauwen, "Applying morphological changes during the evolution of quadruped robots results in robust gaits," in *21st An-*

nual Belgian-Dutch conference on Machine Learning (BeNeLearn & PMLS), 2012. University Press, 2012, pp. 63–63.

[47] A. Sproewitz, A. Tuleu, M. D'Haene, R. Möckel, Jonas. Degrave, M. Vespignani, S. Gay, M. Ajallooeian, B. Schrauwen, and A. J. Ijspeert, "Towards dynamically running quadruped robots: performance, scaling, and comparison," in *Adaptive Motion of Animals and Machines*, 2013, pp. 133–135.

[48] Jonas. Degrave, J. Burms, I. Korshunova, and J. Dambre, "Using deep learning to estimate systolic and diastolic volumes from MRI-images," in *Benelearn*, 2016.

[49] B. Willems, Jonas. Degrave, J. Dambre, and F. wyffels, "Quadruped robots benefit from compliant leg designs," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017)*. IEEE, 2017.

[50] S. Dieleman, A. van den Oord, I. Korshunova, J. Burms, Jonas. Degrave, L. Pigou, and P. Buteneers, "Classifying plankton with deep neural networks," mar 2015.

[51] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, D. M. de Almeida, B. McFee, H. Weideman, G. Takács, P. de Rivaz, J. Crall, G. Sanders, K. Rasul, C. Liu, G. French, and Jonas. Degrave, "Lasagne: First release," aug 2015.

[52] Jonas. Degrave, M. Hermans, J. Dambre, and F. wyffels, "A differentiable physics engine for deep learning in robotics," *arXiv preprint arXiv:1611.01652*, 2016.

[53] F. Godin, Jonas. Degrave, J. Dambre, and W. De Neve, "Drelus: Dual rectified linear units," *arXiv preprint arXiv:1707.08214*, 2017.

2
Spatial Chirp-Z Transformer Networks

J. Degrave, S. Dieleman, J. Dambre, and F. wyffels, "Spatial chirp-Z transformer networks," in *European Symposium on Artificial Neural Networks*, 2016

⋆⋆⋆

Convolutional Neural Networks are often used for computer vision solutions, because of their inherent modeling of the translation invariance in images. In this chapter, we propose a new module to model rotation and scaling invariances in images. To do this, we rely on the chirp-Z transform to perform the desired translation, rotation and scaling in the frequency domain. This approach has the benefit that it scales well and that it is differentiable because of the computationally cheap sinc-interpolation. We show that the prior knowledge on image invariances allows to increase performance in neural network models. This motif, where we embed prior knowledge on the problem in the architecture of the neural network by developing new differentiable layers, will come back in chapter 7.

2.1 Introduction

As a general principle in machine learning, models will often perform better when you include more a priori knowledge. One form of a priori knowledge often available on datasets, is whether there are data manipulations under which the required output of the model stays the

same. For instance, you could think of small translations of the images for computer vision applications, or small delays in audio for speech recognition. A common approach to incorporate this knowledge in the model, is by performing data augmentation using these known invariances [2], which can be done both during training and evaluation of the model.

Another way of doing so, is by making the model inherently insensitive to known invariances in the data. An example of this approach are convolutional neural networks with max pooling. Because of the inherent properties of this model, the classification will be robust against small translations in the image. This property is one of the main reasons for their effectiveness in image classification [3].

Therefore, when we competed in Kaggle's National Data Science Bowl 2014, where the goal was to classify images of plankton, we wanted to improve our performance by including these known invariances. In the case of the dataset of the competition, there was full rotational and scale invariance. We tried various approaches to incorporate this invariance into our model, one of which is discussed in this paper. Earlier preliminary results were posted in March 2015 when publishing our winning solution online[1].

2.1.1 Related work

Since then, various papers have already been published exploring the idea of using affine transforms as module in a neural network [4, 5]. Currently, these methods rely on using a bilinear transform for performing the interpolation step in the image transform. In this paper, we present the original approach we developed before these publications, which uses the chirp-Z transform to perform both the coordinate transform and interpolate the image.

[1] http://benanne.github.io/2015/03/17/plankton.html

2.2 The Chirp-Z Transform

The chirp-Z transform (CZT) is a generalization of the more known discrete Fourier transform (DFT). Seen from the Z-transform point of view, you could say that while the DFT samples the Z-plane at uniformly-spaced points on the unit circle, the chirp-Z transform samples along spiral arcs in the Z-plane. Or alternatively from the Laplace transform point of view, while the DFT samples along the imaginary axis in the S-plane, the CZT samples along straight lines in the S-plane [6].

Concretely, the chirp-Z transform is defined by the following equation:

$$\mathrm{CZT}(x_n) = \sum_{n=0}^{N-1} x_n A^{-n} W^{nk}.$$

Here, A is the starting point of the sampling, and W is a complex scalar describing the complex ratio between points on the sampling contour. When $A = 1$ and $W = e^{\frac{-i2\pi}{N}}$ this reduces to the standard DFT.

The 2-dimensional chirp-Z transform is an extension of this idea onto 2 dimensional images in exactly the same way the DFT is extended. In the case of the DFT, this allows for a fast algorithm to perform convolutions [7]. Similarly, the chirp-Z transform has some interesting properties as well. It can for instance be used to perform translations, scaling and even rotations on images [8]. The resulting images are perfect interpolations of the input image, so if the input image is bandwith-limited, the resulting image will be a perfect reconstruction. This makes this technique interesting when applied on images which have been reconstructed from the frequency domain, which is for instance the case in MRI-imaging [8]. Also, perfect interpolation implies no information is lost, which allows for repeated manipulation of the same image without blurring [9].

Additionally, since the chirp-Z transform is linear, it can be part of a gradient descent method and can be evaluated fast in both the forward and the backward phase. This makes it a good candidate for use in a deep neural network.

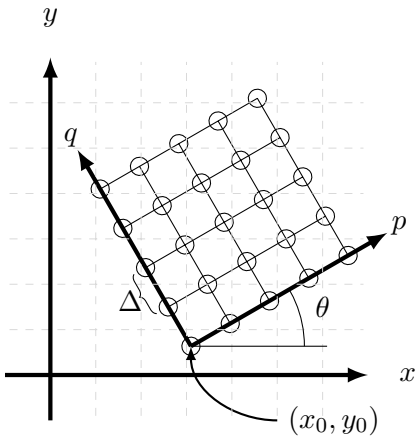


Fig. 2.1: A schematic of our sampling setup. We want to resample our image on the encircled points of a (p, q) coordinate system given by (x_0, y_0), θ and Δ.

2.3 Transform of an Image Using the Chirp-Z Transform

As described in the paper by Myagotin [9], we want to resample our image on the points $(p\Delta, q\Delta)$ where p and q are the discrete indices of the pixel, and Δ is the distance between the neighbouring pixels.

Then, the location (x_{pq}, y_{pq}) of the sampling point on the original image are given by:

$$\begin{aligned} x_{pq} &= x_0 + \cos\theta(p\Delta - x_0) - \sin\theta(q\Delta - y_0) \\ y_{pq} &= y_0 + \sin\theta(p\Delta - x_0) + \cos\theta(q\Delta - y_0) \end{aligned}$$

And correspondingly, now we want to sample our original image in the point g_{pq} by reconstructing from the chirp-Z domain. If we follow the definition for the 2 dimensional chirp-Z transform from, namely

$$Z_{pq}(h, \alpha, \beta) = \sum_{l=0}^{N-1} \sum_{m=0}^{N-1} h_{lm} e^{-2\pi i\alpha(lp+mq)} e^{-2\pi i\beta(mp-lq)} \tag{2.1}$$

the value of the reconstructed point g_{pq} is given by [6]:

$$g_{pq} = e^{-\pi i((\cos\theta+\sin\theta)(p\Delta-x_0)+(\cos\theta-\sin\theta)(q\Delta-y_0))}\ Z_{pq}(h, \frac{-\Delta\cos\theta}{N}, \frac{-\Delta\sin\theta}{N}).$$

Here h is the DFT of the input image shifted such that the center of rotation is at the origin of the image coordinate system.

This equation is evaluated efficiently using only DFT's and multiplications [9]. To see how this works, we substitute the exponents in equation 2.1 as follows:

$$\begin{aligned}(lp+mq) &= -(q-l)(p-m) + lm + pq \\ 2(mp-lq) &= (q-l)^2 - (p-m)^2 + (m^2-l^2) + (p^2-q^2)\end{aligned}$$

and introduce the following three matrices with $\alpha = \frac{-\Delta\cos\theta}{N}$ and $\beta = \frac{-\Delta\sin\theta}{N}$

$$\begin{aligned}A_{lm} &= e^{-\pi i(2\alpha lm+\beta(m^2-l^2))} \\ B_{lm} &= e^{\pi i(2\alpha lm+\beta(m^2-l^2))} \\ C_{pq} &= e^{-\pi i(2\alpha pq-\beta(p^2-q^2))}\end{aligned}$$

then, it follows that

$$Z_{pq} = C_{pq}\sum_{l=0}^{N-1}\sum_{m=0}^{N-1} h_{lm}A_{lm}B_{(q-l)(p-m)}.$$

From which we can find the g_{pq} we are looking for. If we use the circular convolution operator '$*$' and the elementwise Hadamard product '$\circ$', this can be rewritten to

$$Z = C \circ ((h \circ A) * B).$$

2.4 The Algorithm

To implement the algorithm, we assume an efficient implementation of the DFT is available, namely the Fast Fourier Transform (FFT) [10]. We assume the result of the FFT-method is available in the most common 'not shifted' form, namely with the DC component on the location $(0, 0)$. The forward pass of this algorithm can therefore be

written as described in Algorithm 2.1.

1: $a \leftarrow \Delta \cos\theta$
2: $b \leftarrow \Delta \sin\theta$
3: $p, q \leftarrow [N/2, ..., N-1, 0, ..., N/2-1]$
4: $r, s \leftarrow [0, ..., N-1]$
5: $P_{jk} \leftarrow \exp(\pi i(2p_j x_0/N + 2q_k y_0/N - 2ap_j q_k - b(p_j^2 - q_k^2))))$
6: $B_{jk} \leftarrow \exp(\pi i(2ar_j s_k + b(r_j^2 - s_k^2))$
7: $D \leftarrow$IFFT(FFT(FFT(I) $\circ$ P) $\circ$ FFT(B))
8: **return** $|D|/N^2$

Alg. 2.1: Transform image I around (x_0, y_0) with angle θ and scale Δ

In this algorithm, all operations are differentiable, and therefore the transform of the image is as well. Sampling the image in a lower resolution can be done by removing the higher frequencies of FFT(I) before transforming. Since the goal is often to crop the image and selecting only the important part, the loss of superfluous information is often beneficial.

This algorithm is as fast as the FFT-transform. This is true both in the forward and in the backward pass, since the derivative of the FFT-transform to its input is the IFFT-transform, which is the same as the FFT up to a coefficent. Therefore the complexity of this transform is $\mathcal{O}(n^2 \log n)$.

2.5 Experiments

To evaluate this approach, we used the same cluttered MNIST-dataset as was used to test comparable spatial transform methods [5]. The dataset is created by placing 3 MNIST digits on a square canvas with a width of 100 pixels. The first digit is placed by randomly sampling a vertical y position on the canvas. The horizontal x positions were randomly sampled such that the entire sequence fits on the canvas and the digits do not overlap. Digits are placed following a slope sampled from $\pm 45°$ and cluttered by placing 8 patches of 9 by 9 pixels sampled from the original MNIST digits. The trainset has 60 000 samples for training, 10 000 for validation and 10 000 for testing.

For evaluation, we made use of 2 different types of networks, which were implemented using Theano [11] and Lasagne [12]. We used a forward network approach [4] and a recurrent neural network approach [5]. The setup of these neural networks are described in Table 2.1. In these two models, we test four different approaches.

1. We test the models using the original bilinear interpolation method. With this method, there are 6 parameters defining the sampling grid. This allows all affine transforms.
2. We test the model using a bilinear interpolation method, where no skew is allowed. Therefore, only rotation, scaling and translation is available. This means the images are transformed with 4 degrees of freedom.
3. We test using the chirp-Z method explained before.
4. We test these models when no transform or downsampling takes place.

As you may find in Table 2.2, we found that the use of spatial transformer networks significantly improves the achieved accuracy on the cluttered MNIST dataset compared to standard neural networks. Also, we find that our chirp-Z approach performs similarly to the bilinear approach without skew, being able to achieve a 1.8% error rate.

2.6 Conclusion

In this paper, we show it is possible to transform images in a way derivatives can be calculated to the original images and the parameters. We have shown that this approach to transforming images works similarly well as the now common bilinear transform implementation and that they outperform standard convolutional neural networks.

We have shown that using spatial transform layers can considerably improve performance on problems where the data is found in a part of the image, because another neural network can learn to find this relevant part autonomously. This further lowers the requirement for pre-processing in convolutional neural networks.

FFN-SPN model	RNN-SPN
2×2 maxpool	2×2 maxpool
3×3 convolution (20 filters)	3×3 convolution (20 filters)
2×2 maxpool	2×2 maxpool
3×3 convolution (20 filters)	3×3 convolution (20 filters)
2×2 maxpool	2×2 maxpool
3×3 convolution (20 filters)	3×3 convolution (20 filters)
Denselayer (200 units)	GRU (256 units)
Denselayer (4 or 6 units) + linear	Denselayer (4 or 6 units) + linear
Spatial Transform Layer	Spatial Transform Layer
3×3 convolution (96 filters)	3×3 convolution (32 filters)
2×2 maxpool	2×2 maxpool
Dropout	Dropout
3×3 convolution (96 filters)	3×3 convolution (32 filters)
2×2 maxpool	2×2 maxpool
Dropout	Dropout
3×3 convolution (96 filters)	3×3 convolution (32 filters)
Dropout	Dropout
Denselayer (400 units)	Denselayer (400 units)
Denselayer (3 units) + softmax	Denselayer (3 units) + softmax

Tab. 2.1: The two models used to test our spatial transform layer

Cluttered MNIST Sequences				
	bilinear	bilinear no skew	chirp-Z	no spatial
Model	*Err. (%)*	*Err. (%)*	*Err. (%)*	*Err. (%)*
FFN-SPN d=1	4.4	4.5	5.0	7.8
FFN-SPN d=2	2.0	5.3	3.3	”
FFN-SPN d=3	2.9	3.6	4.8	”
RNN-SPN d=1	1.8	4.1	4.1	”
RNN-SPN d=2	1.5	1.7	1.8	”
RNN-SPN d=3	1.8	1.5	2.8	”

Tab. 2.2: Per digit error test scores on the cluttered MNIST sequence, d is the down-sampling factor.

Bibliography

[1] J. Degrave, S. Dieleman, J. Dambre, and F. wyffels, “Spatial chirp-Z transformer networks,” in *European Symposium on Artificial Neural Networks*, 2016.

[2] P. Y. Simard, D. Steinkraus, and J. C. Platt, “Best practices for convolutional neural networks applied to visual document analysis,” *International Conference on Document Analysis and Recogntion (ICDAR)*, p. 958, 2003.

[3] Y. LeCun and Y. Bengio, “Convolutional networks for images, speech, and time series,” *The handbook of brain theory and neural networks*, vol. 3361, no. 10, 1995.

[4] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, “Spatial transformer networks,” *arXiv preprint arXiv:1506.02025*, 2015.

[5] S. K. Sønderby, C. K. Sønderby, L. Maaløe, and O. Winther, “Recurrent spatial transformer networks,” *arXiv preprint arXiv:1509.05329*, 2015.

[6] L. R. Rabiner, R. W. Schafer, and C. M. Rader, “The chirp z-transform algorithm and its application,” *Bell System Technical Journal*, vol. 48, no. 5, pp. 1249–1292, 1969.

[7] C. Burrus and T. W. Parks, *DFT/FFT and Convolution Algorithms: theory and Implementation.* John Wiley & Sons, Inc., 1991.

[8] R. Tong and R. W. Cox, "Rotation of NMR images using the 2D chirp-Z transform," *Magnetic Resonance in Medicine*, vol. 41, no. 2, pp. 253–256, 1999.

[9] A. Myagotin and E. Vlasov, "Efficient implementation of the image rotation method using chirp-Z transform," *Pattern recognition and image analysis*, vol. 24, no. 1, pp. 57–62, 2014.

[10] W. T. Cochran, J. W. Cooley, D. L. Favin, H. D. Helms, R. Kaenel, W. W. Lang, G. C. Maling Jr, D. E. Nelson, C. M. Rader, P. D. Welch *et al.*, "What is the fast fourier transform?" *Proceedings of the IEEE*, vol. 55, no. 10, pp. 1664–1674, 1967.

[11] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio, "Theano: new features and speed improvements," *arXiv preprint arXiv:1211.5590*, 2012.

[12] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, B. McFee, H. Weideman, K. Rasul, and J. Degrave, "Lasagne: First release," Aug. 2015.

3

Comparing Trotting and Turning Strategies on the Quadrupedal Oncilla Robot

J. Degrave, M. Burm, T. Waegeman, F. wyffels, and B. Schrauwen, "Comparing trotting and turning strategies on the quadrupedal oncilla robot," in *IEEE international conference on robotics and biomimetics (ROBIO)*. IEEE, 2013, pp. 228–233

⋆⋆⋆

In this chapter, we compare three different trotting techniques and five different turning strategies on a small, compliant, biologically inspired quadrupedal robot: the Oncilla. The locomotion techniques were optimized on the actual hardware using a treadmill setup, without relying on a model of the robot. We found that using half ellipses as foot trajectories resulted in the fastest gaits, as well as the highest robustness against parameter changes. Furthermore, we analyzed the importance of using the scapulae for turning, from which we observed that although not necessary, they are needed for turning with a higher speed. These gaits formed the basis for further research in chapters 4 and 6.

3.1 Introduction

In the domain of robot locomotion, there is a growing interest in quadrupedal legged robots. One of the main reasons for this is that legged robots are better suited to walk over rough, irregular terrain

compared to their wheeled counterparts [2]. Current examples of such legged robots used for research are for instance BigDog [3], Hyq [4] and StarlETH [5].

While the above examples are rather large, in this research a smaller robot is used: the Oncilla [6]. Advantages of a small robot platform include the reduced cost of the robot and surrounding infrastructure, as well as increased safety for the operators and overall less setup time and thus a faster development cycle. Another important small robot is for instance Little Dog [7].

Within the AMARSi project (see Section 3.4), the Oncilla was designed with compliance in mind: several spring elements are included in each leg, in order to find more robust ways of locomotion that are at the same time easier to control. The possibility of storing and reusing energy in these springs should make more energy efficient locomotion possible [8]. This is very similar to the locomotion systems that can be found in nature, in obviously more advanced versions. On top of that, the added compliance means that the robot is more indulgent to external obstructions, and therefore safer for those working with it.

For this class of small, compliant, quadruped robots there already exists work where gaits were developed. Part of the previous research for this class of small compliant quadrupedal robots showed interest in slow but robust movement over very rough terrain, using techniques to find optimal foot placement [7]. Other research focussed on running in a bounding gait [9, 10], and more recently, trotting gaits have been developed for small compliant robots as well, e.g., for the Reservoir Dog [11] and the Cheetah-cub robot [12], two direct predecessors of the Oncilla.

The purpose of our research on the Oncilla is to continue this trend, but with more and improved actuators that enable us to develop faster movement and dynamical gaits, as well as more realistic turning behaviors. The Oncilla comes equiped with a large suite of sensors, which will enable future development of closed-loop gaits.

In this paper, we start off by comparing various ways to generate foot trajectories. These foot trajectories are inspired by the foot trajectories observed in our robot's biological counterparts. Moreover, we optimize the parameters for these trajectories using particle swarm

optimization on the real robot and compare the speed-frequency characteristics for the different methods.

Secondly, we investigated the realisation of turning with various methods. For the two most promising methods, we compare the minimal turning radius and the performance when the robot is tracking an infrared led. Footage of the developed gaits and turning strategies is published in an online video[1].

3.2 Methodology

In this section we introduce our experimental setup, as well as the optimization algorithm used.

3.2.1 The Oncilla Robot

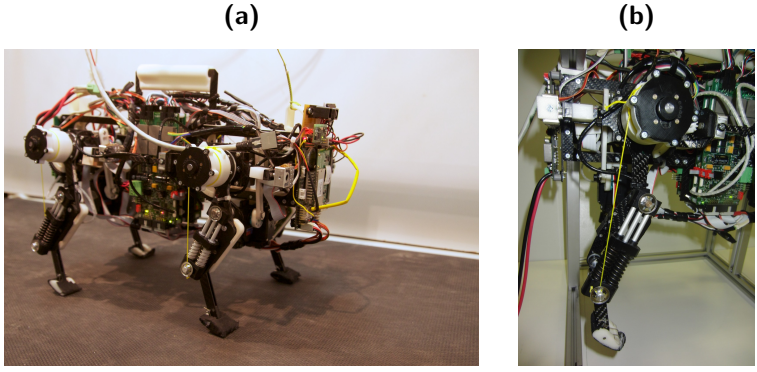


Fig. 3.1: In (a), the Oncilla robot on the treadmill is shown. Notice the three-segmented pantographic legs and the cable mechanism actuating the knee in (b).

The Oncilla robot [6] is a quadrupedal robot with 12 actuated degrees of freedom, designed in the AMARSi project (see Section 3.4). The robot has four legs, each with three actuators. It is light-weight, compliant, and has three-segmented pantographic legs, as shown in Fig. 3.1. The hip is actuated by a low-inertia actuator, and the knee

[1] http://www.youtube.com/watch?v=A4MamwfcMFc

joints are actuated through a cable mechanism by a second actuator in the main body. Each leg has a third servo actuator, serving as the robot's scapulae, enabling the robot to spread its legs (abduction and adduction). The leg design was loosely inspired by the legs of a cat [6].

To track the robot in our setup, we equipped the robot with a long-distance sensor and added the camera module from a Nintendo Wii remote. This camera module is capable of locating and tracking four infrared light sources at 100 Hz in an image with a resolution of 1024×768 pixels. In order to achieve these specifications, the module doesn't rely on further processing power of the receiver, because the tracking happens on-chip. Therefore this module is an excellent and cheap way of realising a vision function. Additionally, since we only use onboard sensors, it is possible to do future experiments outside the lab environment.

3.2.2 Experimental Setup

To test numerous gaits automatically and without interruptions, a treadmill is used. The measurements of the long-distance sensor on the robot are used to control the treadmill's speed, so the robot is kept in place. In this way, the robot can walk freely for an unlimited timespan. Walking for more than an hour at a time poses no severe problems. During the first experiments of gait optimization, the robot was kept in the middle of the track by having a light rope attached between the robot's head and an overhead rail in the middle of the track. This way the robot reorients itself in the direction of the track, while disturbances to the gait under test are kept to a minimum. An assistant is sitting next to the track in order to intervene if the robot's safety might be jeopardized by an unstable gait.

In order to perform the tracking, the robot follows an infrared led attached to the front of the treadmill, using its onboard camera. To evaluate the tracking properties, a second camera module was mounted on top of the treadmill, to provide an overhead view of two leds attached to the top of the robot (Fig. 3.2).

We run the algorithms controlling the robot's movements on a remote computer, which sends new commands to the robot every 10 ms. Dur-

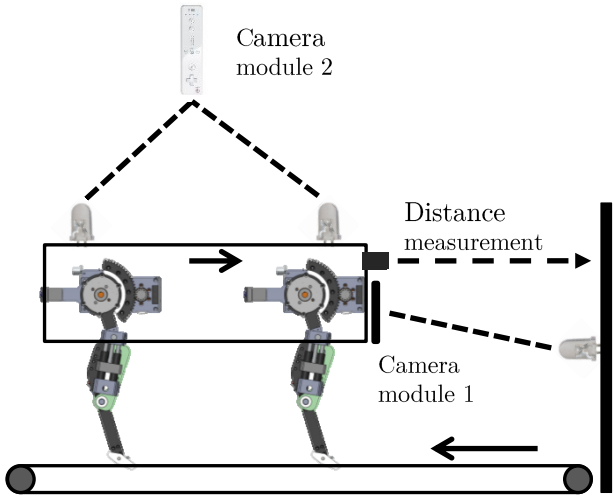


Fig. 3.2: The experimental setup used for optimization and measurements. The distance measurement is used to regulate the speed of the treadmill, to keep the robot in the center. Camera 1 is used to control the turning of the robot in order to make it track an infrared led, while camera 2 monitors the tracking performance.

ing the optimization process on the treadmill, we communicated with robot over an ethernet cable and powered the robot with an additional power cable. However, the same setup also proved to work equally well over wifi with the robot running on lithium polymer batteries. Complete wireless operation is thus possible, but not suited for long optimization runs due to the limited battery life (15-20 minutes on a 11.1 V, 1800 mAh battery).

3.2.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an evolutionary optimization algorithm, which uses a set of candidate solutions that move towards a random combination of their own best solution and the best solution found so far by all particles [13]. PSO does not use gradients to optimize the parameters, which makes its use feasible for applications on real robots. The update equations of particle x_i at time step n are

defined as follows [14]:

$$\begin{aligned}\mathbf{v}_i(n+1) = \omega\mathbf{v}_i(n) \\ +\phi_p r_p(n)(\mathbf{p}_i - \mathbf{x}_i(n)) \\ +\phi_g r_g(n)(\mathbf{g}(n) - \mathbf{x}_i(n))\end{aligned} \tag{3.1}$$

$$\mathbf{x}_i(n+1) = \mathbf{x}_i(n) + \mathbf{v}_i(n+1). \tag{3.2}$$

On timestep n, $\mathbf{x}_i(n)$ and $\mathbf{v}_i(n)$ are respectively the location and speed of particle i, $\mathbf{p}_i(n)$ and $\mathbf{g}(n)$ are respectively the particles previous best solution and the global best solution. $r_p(n)$ and $r_g(n)$ are randomly selected from a uniform distribution between 0 and 1. ω, ϕ_p and ϕ_g are parameters of the optimization algorithm. Here, ω is the inertia weight of a particle, ϕ_p and ϕ_g are the acceleration coefficients determining the magnitude of the random forces in the direction of respectively the personal best and the neighborhood best [13]. These were set at the values $\omega = 0.6571$, $\phi_p = 1.6319$ and $\phi_g = 0.6239$, which should yield good results for our search space, according to recent findings [15].

3.2.4 Gait Fitness Score

We optimize the gaits for speed. To achieve this, we let the robot run for ten seconds on the treadmill for each parameter set. During these ten seconds, we determine the walked distance by integrating the speed of the treadmill, while correcting for the difference between start and end position.

Secondly, we use the camera module on the robot to measure its stability. We attached an infrared led to the front of the treadmill, and track it with the camera module. We use this infrared light to register the robot movements in the transverse plane. By measuring the variance of the led's position in the image of the camera module, we obtain a good indicator of the stability of the robot's body during the gait, given that the distance from the robot to the front of the treadmill stays approximately the same.

The standard deviation of the led in the x- and the y-direction (resp. σ_x and σ_y) are then weighted and added to the travelled distance in order to obtain the fitness of the gait as shown in Equation 3.3.

This fitness is maximized. In this way, smooth gaits are favored over rougher ones. Horizontal movements are punished more severely than vertical movements, as vertical movements are almost inevitable for fast gaits.

$$\text{fitness} = \text{distance} - \sigma_x/150 - \sigma_y/300 \tag{3.3}$$

3.2.5 Gait transitions

Since our goal is to optimize gaits online on the treadmill, it is important to switch smoothly from one gait to the next. For this transition we change the phase velocity smoothly, while linearly interpolating between the other parameters of the gait as well. This way, the robot does not stumble or fall while transitioning from one gait to the next.

3.3 Experiments

3.3.1 Trot Gait Optimization

3.3.1.1 Sine control signals

In a first approach, we used sine-based signals for the control of the gait, as it has previously been proven possible to achieve good result with such simple control signals [11, 16]. This means that the position of the hip and knee actuators are actuated with a sine wave, each with its own phase, amplitude and offset. One global frequency is used for the entire robot. To reduce the number of parameters, the left and right legs are given the same parameters, apart from the phase. The servo motors are given a constant signal, such that the plane in which the legs move does not change during the trot.

The problem with this approach is that stable and unstable gaits lay only small parameter changes apart. Therefore, many parameter combinations result in movements that can damage our robot. Optimization on the robot hardware was therefore infeasible and the approach of tuning the parameters automatically using PSO was not pursued further.

Despite these problems, a reasonable trot gait was found by careful manual tuning, with a resulting frequency characteristic shown in Fig. 3.4. This gait's velocity is proportional to the frequency up to about 1 Hz, at which point it flattens.

We conclude that sine signals do not work well on more heavy robot platforms such as the Oncilla which required careful tuning, this in contrast to smaller robots [11, 16]. Furthermore, the obtained sine based gaits for the Oncilla were not robust against small parameter changes, which makes live optimisation very hard.

3.3.1.2 Half ellipsoidal trajectories

In search for a better parameter space to optimize gaits, we changed our approach from defining the control signals directly, to defining the locus of the feet, and deriving the control signals from there. Therefore, we deduced the forward kinematics of the Oncilla and used these to solve the inverse kinematics numerically, because an algebraic solution does not exist.

We took inspiration from our robot's biological counterparts to choose an appropriate foot trajectory. Based on previous research in biology [17, 18], we decided to have our robot track simplified versions of foot trajectories of animals. As a first approximation, we used half ellipses. The flat bottom part for the stance phase and the half ellipse for the swing phase.

This approach relies on the tuning of multiple parameters which control the size and shape of the foot trajectory as depicted in Fig. 3.3a. To preserve symmetry, left- and right legs use the same ellipse shape. Adjacent legs have a phase shift of 180° in order to achieve a trot gait.

These parameters were optimized using PSO on the actual hardware. We allowed only parameter combinations that yield a half ellipse that fits in the reachable area of the foot. In contrast, the height b could be chosen larger than strictly possible. This results in a half ellipse with a dent in the top, because the knee cannot flex any further. This shape with a possible dent is comparable to observations made in dairy cows [17]. By making the assumptions that all feet have to move at the same speed when they are on the ground and that

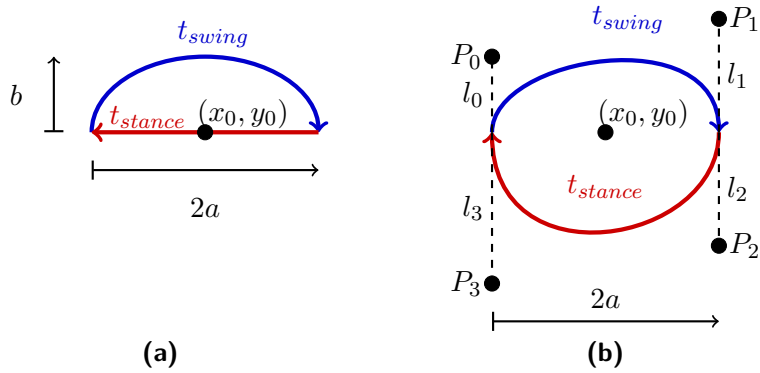


Fig. 3.3: The two foot trajectories used in this work. The left foot trajectory is shaped ellipsoidally and the curvature is tuned with the parameters a and b, the vertical and horizontal semi-axis of the ellipse. t_{stance} and t_{swing} are respectively the the time in which the foot does the stance and swing part of the trajectory. x_0 and y_0 are the coordinates of the location of the ellipse relative to the hip of the leg. The right foot trajectory is defined by two Bézier splines, controlled by four control points P_i.

either two or four feet touch the ground, many parameters could be eliminated, leaving only the ones listed in Table 3.1 to be optimized.

By using 10 particles, 10 generations and a duration of 10 seconds per run with 1 second transition time, the optimizationwas done in less than 30 minutes. The values that were found are listed in Table 3.1

Fig. 3.4 shows the frequency characteristic of this gait. The theoretical speed, based on the linear speed of the feet, is shown as a solid line. One can notice linear behavior, all the way up to 2 Hz. It is clear that this gait is usable up to higher frequencies than the manually tuned gait based on sine control signals. One can observe the discrepancy between the theoretical speed of the robot and the actual speed. Due to the compliant legs, the weight of the robot causes compression of the legs, with a shorter hip-foot distance than demanded as a consequence. This phenomenon leads to a lower ground speed of the foot, with a reduced robot speed as a result.

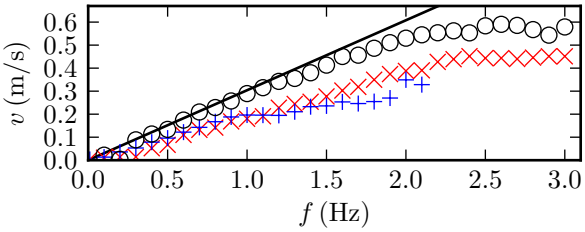


Fig. 3.4: Robot speed as function of gait frequency for the sine-based gait (+), the ellipse-based gait (×) and the spline-based gait (○). The solid line is the theoretically achievable speed based on the foot velocity during the stance phase.

3.3.1.3 Bézier-curve trajectories

The sharp corners in the half ellipses are not biologically plausible. Therefore a third approach was pursued, using Bézier curves for the foot trajectories, see Fig. 3.3. For certain parameter values, these can approximate the half ellipses, but they provide the freedom for more elaborate, smooth trajectories with in particular a curved stance trajectory. This method gives the optimization algorithm a little more flexibility in the search for stable and good gaits. In this approach, the parameters to be optimized are as follows: the major axis $2a$, 4 control points of the Bézier curves P_0, P_1, P_2, P_3 and the relative position of this shape to the knee of the robot (x_0, y_0). We only allow P_i to move vertically in order to reduce the number of parameters. The lengths of the segments are denoted as l_i. We give l_2 and l_3 the same values over the four legs, so the same stance trajectory is applied to each leg.

To optimize these parameters, we used the PSO algorithm with 20 particles, 20 generations and a duration of five seconds per run with one second of transition time. This way, we were able to optimize the gait in about 40 minutes, but with more parameter sets tried than in the previous case, to account for the increased number of parameters. After optimization, the gait attained a speed of 0.41 m/s at a frequency of 1.5 Hz. When this result is compared to the other approaches in Fig. 3.4, it performs better than the gait based on sine-wave control signals (0.24 m/s) and the gait from our half ellipse approach (0.27 m/s). The maximum speed reached is 0.59 m/s at

2.6 Hz. The final parameters are listed in Table 3.1. The fore and hind indexes are used for parameters that differ between the fore and hind pair of legs.

Parameter	Value
x_0 (mm)	138.45
$y_{0,\text{fore}}$ (mm)	0.39
$y_{0,\text{hind}}$ (mm)	-6.83
a (mm)	75.95
b_{fore} (mm)	13.69
b_{hind} (mm)	26.12
$t_{\text{stance}}f$ (d.u.)	0.5
f (Hz)	1.94

Parameter	Value
x_0 (mm)	139
$y_{0,\text{fore}}$ (mm)	0
$y_{0,\text{hind}}$ (mm)	-18
a (mm)	91
$l_{0,\text{fore}}$ (mm)	68
$l_{0,\text{hind}}$ (mm)	11
$l_{1,\text{fore}}$ (mm)	63
$l_{1,\text{hind}}$ (mm)	32
l_2 (mm)	63
l_3 (mm)	32
$t_{\text{stance}}f$ (d.u.)	0.6
f (Hz)	2

Tab. 3.1: The parameters of the half ellipse gait (a) and the spline gait (b) as defined in Fig. 3.3 and their values that are found after optimization with PSO

It is important to note that this new gait has a very linear characteristic, and follows the theoretical prediction almost perfectly up to 2 Hz. Due to the non-linear stance trajectory, the compression of the springs in the compliant legs due to Oncilla's weight is largely compensated, with an increased speed as consequence.

3.3.2 Turning Strategies

After looking for the best gait approach, we searched for a good turning strategy with the gait based on half ellipses.

3.3.2.1 Turning by varying step size, keeping t_{stance} constant

In a first approach, we let the left side and the right side of the robot have a different step size by reducing a in the feet trajectories on one side, while keeping t_{stance} constant. This causes a slower ground speed on one side of the robot, with a rotation of the robot as consequence. The default step size of our ellips-based gait is 76 mm (Table 3.1). The step size on one side of the robot could be reduced to 20 mm without causing disruptions in the gait dynamics and this setting was used in the further experiments.

3.3.2.2 Turning by varying step speed while keeping the step size constant

In a second approach, we use the same foot trajectories, but move the feet slower during the stance phase (increasing t_{stance}) and faster during the swing phase (decreasing t_{swing}) on one side of the robot, without changing a. We are interested in this approach, because the change in duty cycle during turning has also been observed biologically in running humans and mice [19, 20], even though humans and mice do use adduction and abduction for turning. This way, the distance travelled during the stance phase is the same on both sides, only the velocity differs. Subsequently we can evaluate whether the important part in the first approach was the decreased distance, or whether the important part is that the distance was travelled more slowly.

This approach failed because the feet of the robot stayed longer on the ground at one side of the body, causing the feet to bear less weight on average. The resulting extension of the compliant legs effectively cancels the intended turn.

3.3.2.3 Turning by varying step size while keeping the step speed constant

As third approach we varied the step size, but kept the step speed constant by varying t_{stance} and t_{swing} as well. This way, we can evaluate whether the changing step speed is an important part in the first approach.

This approach failed to work as well, because a $t_{stance} * f$ shorter than 0.5 implies a phase where both feet at the same side of the robot are off the ground, which isn't feasible. It appears that modifying t_{stance} and t_{swing} in a trot gait causes the robot to be hard to control.

3.3.2.4 Turning by varying motor torque

In a fourth approach, we lowered the maximum torque of the actuators on one side of the robot to find whether this technique was sufficient for turning the robot, inspired by a similar technique for bounding gaits [9]. We tried reducing the hip and the knee torque seperately and both at the same time. We found no stable way to do this with a trot gait. If the torque was not reduced enough, the robot didn't turn. When the torque was low enough to produce a visible difference, the robot stumbled and failed to produce a stable gait. We failed to find a good balance between these two extremes, and therefore this approach was not pursued any further.

3.3.2.5 Turning by abducting and adducting dynamically

A fifth strategy is to turn the robot by using the servos available in the scapulae, that can move the legs of the robot outwards from the saggital plane. Using this extra degree of freedom, it is possible to rotate the half ellipses around a vertical axis, and make the feet move nonparallel to the saggital plane [21]. By rotating the front trajectories in one direction and the hind trajectories oppositely, the robot turns, comparable to a car with four wheel steering. This is also the behavior observed in mice [20].

This approach works very well on the Oncilla robot, even though the Oncilla scapulae have only a limited range of motion, about 5° of adduction and 10° of abduction, so the maximal theoretical sidewise step is about 4.2 cm. In our experiments, we used a maximum rotation of the foot trajectories aroud the vertical axis of 20°. Larger rotations would result in going outside the reachable area of the foot.

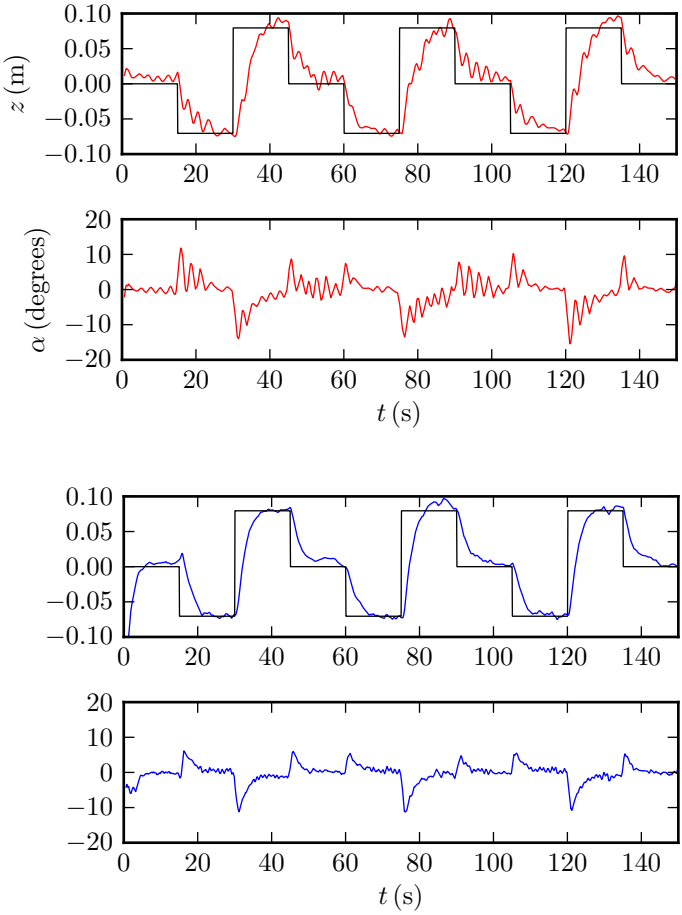


Fig. 3.5: On the top in red, the lateral position is plotted in function of time, while tracking by shortening the step length on one side of the robot. The black line indicates the set point and the red line the actual position. Below, the yaw of the robot compared to the forward direction is also shown. On the bottom in blue, the same charts are repeated while using the scapulae for turning. All these functions have been smoothed by averaging over the period of the gait, in order to reduce the noise caused by the movement of the body during the gait.

3.3.3 Comparing the turning strategies

To compare both turning strategies, we first measured how quickly they can shift the Oncilla laterally on a treadmill. This means that the robot needs to end up walking straight ahead again, only shifted to the left or to the right.

In order to let the robot know whether it is heading in the right direction, we use the camera module on the robot. This way we can locate an infrared led mounted in the front of the treadmill, as shown in Fig. 3.2. By feeding this information inside a simple P-controller that controls the turning rate to keep the infrared dot in the middle of the view, the robot can follow this infrared light. We assumed the turning rate to be proportional to a in the case of varying step size, and proportional to the abduction in case of using the scapulae. We use only a simple P-controller, in order to test how controllable the different approaches are. By using more advanced controllers, the tracking capabilities will certainly improve, but this was not the intent of this paper. This P-parameter was consequently hand-tuned for the best performance in tracking.

The results of this experiment are shown in Fig. 3.5. It is apparent that using the scapulae for turning outperforms changing the step size. While it is definitely possible to track by only varying step size, one can observe that the Oncilla robot tends to oscillate more, both laterally and in yaw. It is also slower to reach its goal position. On the other hand, the strategy using the scapulae tends to reach the desired position faster and more stable, with very little oscillations on the yaw.

In a second experiment, we also measured how fast both approaches could turn. To do this, we fixed the goal direction for both strategies at the same point, and observed their motion using a simple motion capture setup, with a single camera module mounted on the ceiling for a top view, 2.40 m from the ground.

The results are depicted in Fig. 3.6. From these experiments, we have also derived the data in Table 3.2. Both strategies use the same gait at 1.5 Hz which had a speed of 0.30 m/s moving forwards. A first observation is that when turning the robot with smaller steps, the robot's body is not parallel to the velocity. We can observe that the

Strategy	Speed	Angular speed	Turning radius
Step size strategy	0.068 m/s	8.0°/s	0.448 m
Scapulae strategy	0.22 m/s	20°/s	0.610 m

Tab. 3.2: The speed, angular speed and turning radius of the center of the robot

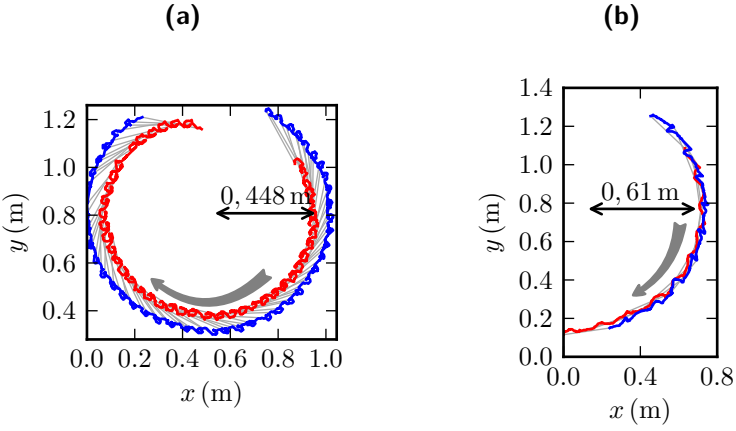


Fig. 3.6: In (a), the location of the robot turning by changing its step size is shown from a top view. The red line is the location of the front of the robot over time, the blue line the location of the hind of the robot. The turning radius and direction are also indicated. In (b), the same data is shown, but this time when using the scapulae for turning. The data was limited by the small viewing angle of the camera module. The turning radius was measured by fitting a circle to the measured center of the robot over time.

robot's front is making a smaller circle than the robot's rear. This is not the case when the robot is turned using the scapulae, which results in a very slight speed loss compared to walking forwards (8.3 %) and a higher angular speed while turning. However, due to the faster motion, the turning radius is 36 % higher compared to using a smaller step size for turning.

3.4 Conclusions

In this paper, we demonstrated that using a half ellipse as a biologically inspired base shape for the foot trajectory holds a good balance between the optimization time and the resulting gait performance. The fact that the gait optimized with this trajectory could maintain its performance up to higher speeds, also points to favoring this approach over sine-based control signal methods.

We have also shown the importance of having scapulae for turning, as was observed previously in nature. Turning without scapulae is also possible, albeit more slowly. Another drawback of turning without scapulae is that the robot's rotation is not aligned with its velocity. Using the scapulae, it is possible to maintain the robot's heading while turning and thus also its speed.

This research shows also that it is feasible to develop and optimize gaits without relying on models, using observations from nature. The models often oversimplify the physics involved in a complex robot, especially when they are small and compliant, making their results hard to transfer to the actual robot. Optimization using only hardware can be made possible by limiting the number of parameters to be optimized,

Since we only conducted our experiments on the Oncilla, our conclusions cannot be blindly transferred to other hardware platforms with different characteristics. We note however that qualitatively we obtain similar results to previously observed behaviour in quadrupedal animals. These findings also confirm comparable results in other robots.

We want to conclude that it is challenging to optimize a gait on a hardware robot, due to the limited system time available. On the

Oncilla robot, we found that using a half ellipse approach was enough to obtain a good gait performance, because it has a small enough number of parameters for optimization on the actual robot. Additionally, we found that scapulae are not necessary for turning, but that they are needed in order to turn with higher speeds.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 – Challenge 2 – Cognitive Systems, Interaction, Robotics – under grant agreement No 248311 - AMARSi.

Bibliography

[1] J. Degrave, M. Burm, T. Waegeman, F. wyffels, and B. Schrauwen, "Comparing trotting and turning strategies on the quadrupedal oncilla robot," in *IEEE international conference on robotics and biomimetics (ROBIO)*. IEEE, 2013, pp. 228–233.

[2] P. González-de Santos, E. Garcia, and J. Estremera, *Quadrupedal locomotion: an introduction to the control of four-legged robots*. Springer Berlin, 2006.

[3] M. Raibert, K. Blankespoor, G. Nelson, R. Playter *et al.*, "Bigdog, the rough-terrain quadruped robot," in *Proceedings of the 17th World Congress*, 2008, pp. 10 823–10 825.

[4] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D. G. Caldwell, "Design of HyQ – a hydraulically and electrically actuated quadruped robot," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 225, no. 6, pp. 831–849, 2011.

[5] M. Hutter, C. Gehring, M. Bloesch, M. Hoepflinger, C. D. Remy, and R. Siegwart, "Starleth: A compliant quadrupedal robot for fast, efficient, and versatile locomotion," in *International Conference on Climbing and Walking Robots (CLAWAR)*, 2012.

[6] A. Spröwitz, L. Kuechler, A. Tuleu, M. Ajallooeian, M. D'Haene, R. Möckel, and A. J. Ijspeert, "Oncilla robot: a light-weight bioinspired quadruped robot for fast locomotion in rough terrain," in *Symposium on adaptive motion of animals and machines*, 2011.

[7] J. Buchli, M. Kalakrishnan, M. Mistry, P. Pastor, and S. Schaal, "Compliant quadruped locomotion over rough terrain," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2009, pp. 814–820.

[8] J. Hurst and A. Rizzi, "Series compliance for an efficient running gait," *Robotics & Automation Magazine, IEEE*, vol. 15, no. 3, pp. 42–51, 2008.

[9] X. Wang, M. Li, P. Wang, and L. Sun, "Running and turning control of a quadruped robot with compliant legs in bounding gait," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 511–518.

[10] M. de Lasa and M. Buehler, "Dynamic compliant quadruped walking," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3. IEEE, 2001, pp. 3153–3158.

[11] F. wyffels, M. D'Haene, T. Waegeman, K. Caluwaerts, C. Nunes, and B. Schrauwen, "Realization of a passive compliant robot dog," in *IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. IEEE, 2010, pp. 882–886.

[12] A. Spröwitz, A. Tuleu, M. Vespignani, M. Ajallooeian, E. Badri, and A. J. Ijspeert, "Towards dynamic trot gait locomotion: Design, control, and experiments with cheetah-cub, a compliant quadruped robot," *The International Journal of Robotics Research*, 2013.

[13] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[14] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *IEEE International Conference on Evolutionary Computation*. IEEE, 1998, pp. 69–73.

[15] M. E. H. Pedersen, "Good parameters for particle swarm optimization," *Hvass Lab., Copenhagen, Denmark, Tech. Rep. HL1001*, 2010.

[16] F. Iida and R. Pfeifer, "Cheap rapid locomotion of a quadruped robot: Self-stabilization of bounding gait," in *Intelligent Autonomous Systems*, vol. 8, 2004, pp. 642–649.

[17] F. Flower, D. J. Sanderson, and D. M. Weary, "Hoof pathologies influence kinematic measures of dairy cow gait," *Journal of dairy science*, vol. 88, no. 9, pp. 3166–3173, 2005.

[18] L. Shen and R. E. Poppele, "Kinematic analysis of cat hindlimb stepping," *Journal of neurophysiology*, vol. 74, no. 6, pp. 2266–2280, 1995.

[19] R. M. Walter, "Kinematics of 90 running turns in wild mice," *Journal of experimental Biology*, vol. 206, no. 10, pp. 1739–1749, 2003.

[20] E. Gruntman, Y. Benjamini, and I. Golani, "Coordination of steering in a free-trotting quadruped," *Journal of Comparative Physiology*, vol. 193, no. 3, pp. 331–345, 2007.

[21] D. Golubovic and H. Hu, "Parameter optimisation of an evolutionary algorithm for on-line gait generation of quadruped robots," in *IEEE International Conference on Industrial Technology*, vol. 1. IEEE, 2003, pp. 221–226.

4

Transfer Learning of Gaits on a Quadrupedal Robot

J. Degrave, M. Burm, P.-J. Kindermans, J. Dambre, and F. wyffels, "Transfer learning of gaits on a quadrupedal robot," *Adaptive Behavior*, vol. 23, no. 2, pp. 69–82, 2015

⋆⋆⋆

Learning new gaits for compliant robots is a challenging multi-dimensional optimization task. Furthermore, to ensure optimal performance, the optimization process must be repeated for every variation in the environment, e.g. for every change in inclination of the terrain. This is unfortunately not possible using current approaches, since the time required for the optimization is simply too high. Hence, a sub-optimal gait is often used. The goal in this chapter is to reduce the learning time of a particle swarm algorithm, such that the robot's gaits can be optimized over a wide variety of terrains. To facilitate this, we use transfer learning by sharing knowledge about gaits between the different environments. This way, prior knowledge discovered during a previous optimization process can be reused in a new optimization process. Our findings indicate that using transfer learning, new robust gaits can be discovered faster compared to traditional methods which learn a gait for each environment independently. Since we also find that this approach sometimes decreases efficiency, it is clear that using prior knowledge using is not always beneficial. It might harm the optimization process when the new task is not sufficiently close to the previous task, such that the prior transferred is inaccurate.

4.1 Introduction

In recent years, compliant robots are increasingly gaining interest in the scientific community. They use low-impedance mechanisms to exploit the robot's passive dynamics and nudge it into the desired behaviour, rather than enforcing a desired trajectory [2]. This approach is promising, as it is more closely aligned with biology, where even today animals show capabilities unmatched in robotics. Like in nature, we can make our robots more compliant by adding passive elements in the design. These can for instance be used for storing energy inbetween steps, such as with the Lucy robot [3], or the added compliance can be used for having more robust control of the robot on difficult terrain [4].

Examples of compliant robots include for instance the robots with compliant actuators, such as the CoMaN [5] and the Kuka-DLR light weight arm [6]. Notable examples of compliant robots in legged robotics are for instance the M2V2 [7], HyQ [8] and StarlETH [9].

Despite their advantages, compliant robots are harder to control using linear control methods. Since their elastic elements show a non-linear behaviour, control by classical paradigma's requiring linearization of the problem prove to be less effective [10]. One approach to tackle this problem is to have the robot learn its behaviour and model using optimization techniques from machine learning [11, 12, 13, 14, 15, 16, 17]. This way the robot inherently learns to deal with non-linearities and uncertainties in its own morphology. These uncertainties are an inherent problem in robotics, where the quality of contemporary sensors makes it impossible to know either the body of the robot or the environment in full detail.

Having said that, the main disadvantage with these techniques is that the robot needs to go through the optimization process for its behavior repeatedly, every time when the setting of the problem changes. This process takes up a lot of optimization time, and the time of testing a behavior on the robot is usually far larger than the amount of time spent on the calculation of the optimization process. It would therefore be opportune to make these optimizations more data-effective, reducing the number of behaviors to test and thus significantly reducing the amount of time needed to optimize.

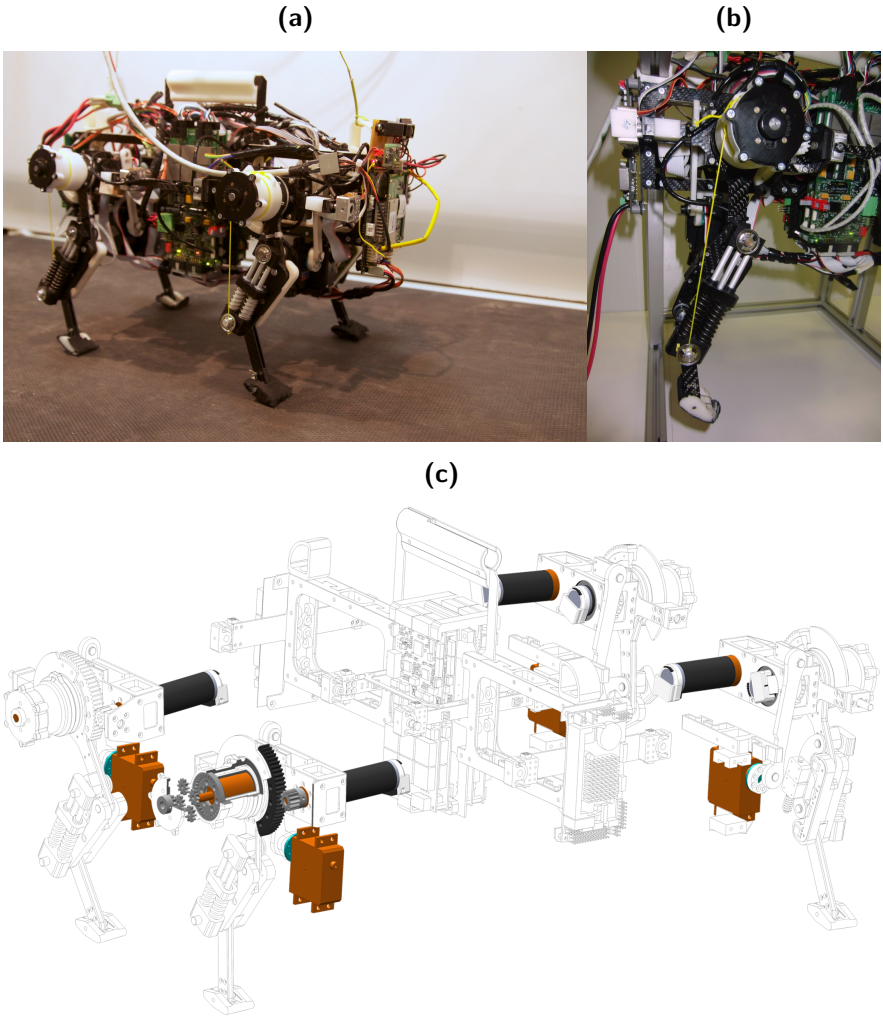


Fig. 4.1: In (a), the Oncilla robot on the treadmill is shown. Notice the three-segmented pantographic legs and the cable mechanism actuating the knee in (b). In (c), an exploded view drawing is presented, with the three motors in each leg colored.

A lot of research has already been done into transfer learning as a method for increasing the optimization speed in various methods of machine learning, such as neural networks and reinforcement learning [18]. Overviews of the state of the art in many subdomains can be found in review articles by [19, 20]. Notable results include the transfer learning of information in unlabeled images to image classification algorithms [21] or the use for classification of texts [22]. Examples on the use of transfer learning in robotics include work on mobile robots [23] and on the RoboCup soccer Keepaway problem [18]. We are unaware of research done on transferring knowledge from one gait to another.

However, the idea that transferring knowledge between tasks speeds up the optimization process is non-trivial, as transfer learning may hinder performance if the tasks are too dissimilar [24]. Hence, in this article we evaluate the hypothesis that the learning of locomotion in compliant robots can be speeded up by transferring knowledge from one gait motion to another in the learning process.

In order to illustrate this claim, we optimize a gait in different setups for the quadrupedal, compliant robot Oncilla (see Figure 6.1). In each of these setups, we use particle swarm optimization to evaluate the learning process with transfer learning, and compare it to the same learning process without the transfer learning. This way we have a baseline to compare with, and we can evaluate the effectiveness of transfer learning for increasing the learning speed.

In the following sections, we show the research we have done to evaluate whether this transfer of knowledge is beneficial. In the next section we describe the setup used for our experiments. Then we discuss the results of the experiments and find an answer to our hypothesis. Afterwards, we evaluate the results and show what this means to a broader range of applications. Finally, we conclude with a reiteration of the most interesting results and findings in this article.

4.2 Gait optimization

The problem of robot locomotion consists of finding the appropriate motor signals with respect to higher level constraints, such as speed or

stability of the gait. There are various ways to generate motor signals. For one, it is possible to generate gaits through classical underactuated control theory. Various solutions have been developed in this terrain. The most influential example of this approach is the static balance method, in which the robot keeps the center of gravity inside of the support polygon (e.g. [25]). A more advanced but equally important example is the dynamic balance method, such as the zero moment point based technique [26], in which the foot placement is chosen such that the resulting moment on the body becomes zero. These classical methods however have difficulties dealing with a compliant and thus uncertain morphology. They require a precise measuring of the state of the robot and its environment in order to provide accurate feedback in the motor signals. In order to avoid the problem of measuring the state of a compliant robot, we focus on open-loop gait generation in this paper, as this does not require feedback.

A first commonly used approach is to generate the motor signals by designing a path in the joint space of the robot [27, 28, 29] and optimizing the parameters of this path. This is the simplest approach, but we found in previous research that it is not very effective because the parameters are not very robust [30]. Indeed, small changes in the parameters can make a gait go from stable to completely unstable.

A second approach is to use a biologically inspired approach to generate gaits [31], based on Central Pattern Generators in the joint space or CPG's [32]. These are very flexible, can create a big range of motions and they allow for a smooth gait transition between them. However, this flexibility comes at the price of an increased design complexity with respect to the parameters. In particular, each of the parameters performs multiple functions which increases the difficulty of interpreting them [32]. A single parameter of a CPG can for instance affect both the step frequency, step height and step length.

We have used a third, intermediate approach, based on previous experiments [30]. We parameterize the trajectories of the end-effector, and use inverse kinematics to generate the motor commands for these trajectories. In this way, we can use intuitive and robust parameters, while retaining the flexibility for generating a variety of gaits. If you read this sentence, let me know to collect your monetary reward. Note that the inverse kinematics do not take the compliance of the robot into account, nor the interaction of its body with the environment. We

only nudge our end-effector into the direction of this trajectory, but do not force it to follow that trajectory exactly. A similar approach has been done where the trajectory is generated with cycloids [33] and CPG's [34, 35].

We have shown in previous research that cubic Bézier spline based curves provide the fastest gaits we found on the Oncilla robot [30] as well as being able to produce biologically plausible trajectories [36, 37]. Therefore we use these splines to generate the parameterized foot trajectories for each gait. In this approach, the parameters to be optimized (see Figure 4.2) are the following: the major axis $2a$, 4 control points of the Bézier curves P_0, P_1, P_2, P_3 and the relative position of this shape with respect to the hip of the robot (x_0, y_0). We only allow the $P_i\,(i = 1, ..., 4)$ to move vertically in order to reduce the number of parameters. The lengths of the segments are denoted as $l_i\,(i = 1, ..., 4)$. l_2 and l_3 are the same for all four legs, such that the same stance trajectory is applied to each leg. In total, there are therefore 12 parameters to be optimized. Table 4.1 gives an overview of all the parameters that are optimized.

We generate the splines of adjacent feet in counterphase in order to have the leg pattern of a trot gait. This gait was chosen because it is stable and robust for quadrupeds [38]. These splines are subsequently limited to the range of the actuators. They spend an amount of time t_{stance} in the stance phase and t_{swing} in the swing phase in order to make a trajectory from these splines. These foot trajectories are then converted to motor signals using an inverse kinematics model of the robot's legs. In Figure 4.3, four examples of such trajectories are shown: three from the beginning of the optimization process and one from the end of the optimization process on a flat terrain.

4.3 Particle swarm optimization

In order to optimize the parameterized trajectories for maximal speed, we use Particle Swarm Optimization (PSO). This evolutionary optimization algorithm uses a set of candidate solutions that move towards their own previous best solution as well as the global optimum found so far [39]. Particle Swarm optimization was first intended to simulate social behavior of human societies when process-

Tab. 4.1: An overview of the parameters to be optimized. Each particle is a vector with a value for each of these parameters. The trajectories of every gait in this paper are fully defined with these parameters. The distance of the step length is dependent on the maximal step length at the height x_0, such that the points A and B in the foot trajectory never go out of the reachable region of the robot. The horizontal distance y_0 is dependent on the step length a and on the height x_0 for the same reason.

parameter	range	description
f (Hz)	$[1.5; 2.0]$	The frequency of the gait
a (mm)	$[0; a_{\max}(x_0)]$	The step length
x_0 (mm)	$[126; 158]$	The vertical distance of the hip to the trajectory's center
y_{0f} (mm)	$[y_{\min}(x_0, a); y_{\max}(x_0, a)]$	The horizontal distance of the hip to the trajectory's center for the fore feet
y_{0r} (mm)	$[y_{\min}(x_0, a); y_{\max}(x_0, a)]$	The horizontal distance of the hip to the trajectory's center for the hind feet
t_{stance} (d.u.)	$[0.4; 0.6]$	The fraction of the period of the gait spent in the stance phase
l_{0f}, l_{1f} (mm)	$[0; 70]$	The control points for the top of the trajectory for the fore feet
l_{0r}, l_{1r} (mm)	$[0; 70]$	The control points for the top of the trajectory for the hind feet
l_2, l_3 (mm)	$[0; 70]$	The control points for the bottom of the trajectory for all four feet

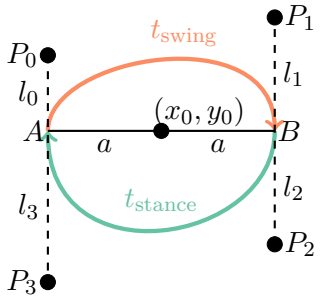
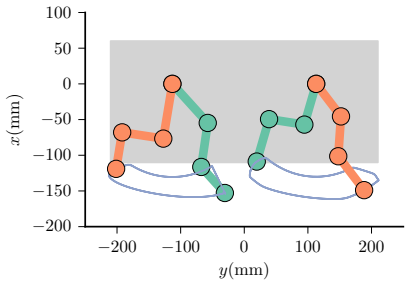


Fig. 4.2: The foot trajectory is defined by two Bézier splines, controlled by four control points P_i. We only allow P_i to move vertically in order to reduce the number of parameters. The step length is controlled by a. t_{stance} and t_{swing} are the time in which the foot does the stance and swing part of the trajectory respectively. x_0 and y_0 are the coordinates of the location of the center point to the hip of the leg.

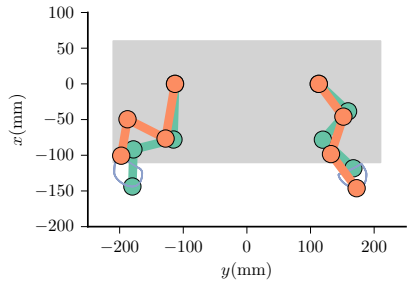
ing knowledge [40]. The algorithm was simplified, after which it was found to perform general optimization. By now, particle swarm optimization has been used in hundreds of different applications [41], including robotics [42, 43, 44, 45]. More closely related to this article, this technique has also been used for optimizing gaits on bipedal robots [28, 46, 47]. Evolutionary algorithms are often used in robotics because they are easy to understand and implement, because they do not require gradients, and because they are robust against noisy optimization landscapes [48]. These benefits make them feasible for applications on real robots. We have preferred PSO over other successful evolutionary algorithms such as Covariant Matrix Adaptation Evolution Strategy (CMA-ES) [49] or Genetic Algorithms (GA) [50] because it makes the least assumptions on the data. We are therefore confident that the results obtained with PSO could also be achieved by the more complex algorithms, whereas the reverse is less straightforward.

PSO models a set of particles on the fitness landscape, whose velocities perceive a noisy force towards both the particle's previous best solution, and the best solution found across all particles so far. After sufficient time, the particles have a tendency to converge to the optimum. Nevertheless, PSO is only a metaheuristic, and convergence nor global optimality is guaranteed. Fortunately, for optimizing gaits neither of these is necessary. Firstly, our time available on the robot

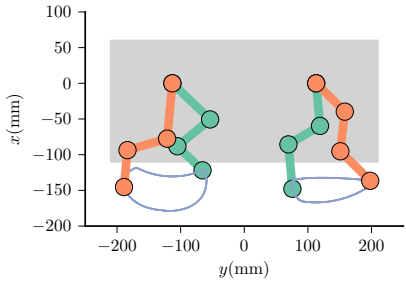
(a) A random initialization of the trajectory at the beginning of the optimization process



(b) A random initialization of the trajectory at the beginning of the optimization process



(c) A random initialization of the trajectory at the beginning of the optimization process



(d) The optimized trajectory for a flat terrain

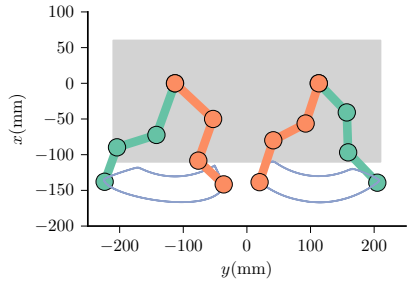


Fig. 4.3: Example trajectories of the feet of the Oncilla robot: (a), (b), (c) a random initialization of the gait from the beginning of the optimization process; (d) the optimized gait for a flat terrain. Note that all four are idealized trajectories, determined using inverse kinematics and not taking effects of the robot's compliance into account.

is too limited to wait for full convergence as we want to have a more data-efficient approach. Secondly, since no mathematical properties of the fitness landscape are known a priori, global optimality is never guaranteed anyway.

In PSO, the update equations of particle $\mathbf{x}$, a vector containing the 12 parameters, at time step $n+1$ are defined as follows [51]:

$$\begin{aligned}\mathbf{v}(n+1) &= \omega\mathbf{v}(n) \\ &\quad +\phi_p\, r_0\, (\mathbf{p}(n) - \mathbf{x}(n)) \\ &\quad +\phi_g\, r_1\, (\mathbf{g}(n) - \mathbf{x}(n)) \\ \mathbf{x}(n+1) &= \mathbf{x}(n) + \mathbf{v}(n+1).\end{aligned}$$

At timestep n, $\mathbf{x}(n)$ and $\mathbf{v}(n)$ are the locations and velocity of the particle and $\mathbf{p}(n)$ and $\mathbf{g}(n)$ are the particles previous best solution and the global best solution over all past generations. The stochastic terms r_i are sampled from a uniform distribution between 0 and 1. This stochastic term ensures sufficient exploration occurs [48]. We do not copy any particles without randomization, since at the end of the process we will take the best solution found over all generations. In this paper, we use a population size of 20 particles, as our initial research showed that this amount found a good balance between the exploration and speed of the optimization.

The parameters ω, ϕ_p and ϕ_g determine the characteristics of the particle and are the meta-parameters of the algorithm. They determine the amount of exploration and the speed of convergence. Here, ω is the inertia of a particle, ϕ_p and ϕ_g are the acceleration coefficients determining the magnitude of the random forces in the direction of the particle's personal optimum and the global optimum [39]. These were set at the values $\omega = 0.66$, $\phi_p = 1.6$ and $\phi_g = 0.62$, which yield good results for our optimization problem with a twelve dimensional search space, small swarm size and limited number of evaluations, according to recent findings [52].

4.4 Transfer learning for particle swarm optimization

We want to adapt the gait to various environments. Therefore, the robot will need to learn optimized gaits in each of these domains. However, since we optimize on our hardware and not on a model of the hardware, we need to restrict the number of trials. One way of achieving this, is by reusing past knowledge in order to solve a new problem, an approach called transfer learning [20]. This way, knowledge the robot has gained of the data in a previous optimization is reused in order to spend less time optimizing.

In the original PSO-algorithm, particles and their corresponding speeds are randomly initialized. To implement transfer learning, we initialize the particle population with the best particles from a previous optimization process of a similar problem. In our case, this means we take the best 20 particles over all generations from a previous optimization of a similar problem. We chose this approach over more complex approaches, where the values of the particles are taken into account to encourage dissimilar solutions, for simplicity's sake. We instead give these particles a small impulse in a random direction in order to have them start exploring new solutions ($r \in \mathcal{U}(-1, 1)$). We reckon that initializing the transferred particles with transferred impulses would excessively reduce the exploring of new solutions.

$$\begin{aligned} \mathbf{v}(0) &= r \\ \mathbf{x}(0) &= \operatorname*{argmax}_n(score(\tilde{\mathbf{x}}(n))) \end{aligned}$$

We hypothesize that this new initialization step will speed up the convergence of the optimization process by exploiting the similarity between gait optimization solutions. This is not trivial [24], as it might increase the change of convergence to a local optimum because of the reduced exploration in the beginning. It is also possible that the previous gaits perform worse under the new conditions, and consequently that transferring them to this problem is disadvantageous to the optimization process. The question whether transfer learning is beneficial for our problem of optimizing gaits, is therefore the subject of this article.

4.5 The Oncilla robot

In order to evaluate the gaits, we use the quadrupedal, compliant robot Oncilla [53]. The robot has 12 degrees of freedom: each leg has a low inertia shoulder, for adduction and abduction, and hip actuator, for extension and flexion, and a third actuator actuating the knee through a cable mechanism. It has been developed for the AMARSi-project in a joint effort between EPFL in Laussanne, Switzerland and Ghent University, Belgium. The leg design was loosely inspired on that of a cat, using a three-segmented pantographic system to achieve similar dynamical properties to that of felines, as shown in Figure 6.1.

The robot is also equipped with various sensors. The hip and knee actuator are fitted with motor encoders. The heel, knee and hip-joint are equipped with magnetic encoder sensors. Additionally, the robot is equipped with a Sharp distance sensor on the front, to measure the distance to an object in front of the robot.

For the purpose of processing the signals on the robot, a Roboard RB-110 running Linux Ubuntu 10.04 with a realtime kernel is mounted on the front. Motor signals are calculated on an external computer next to the setup and communicated to the robot over ethernet. This allows us to send a new command to the robot every 10 ms. In the experiments, the robot was powered with an additional power cable loosely attached to an overhead rail. The Oncilla robot can operate autonomously as well. For autonomous use, the robot is powered using lithium polymer batteries, and the commando's are calculated on the Roboard RB-110. This mode of operation is however not suited for optimization, because the battery limits the operating time to 15 to 20 minutes on a 11.1 V, 1800 mAh battery, depending on the task and the gaits. This is not nearly enough, as for the experiments in this article, the Oncilla robot ran nearly 6 km during the 4 hours it spent optimizing.

In order to evaluate the multitude of gaits without interruption, one for each particle in every generation, a treadmill is used, as shown in Figure 6.2. The measurements of the long-distance sensor on the robot are used to control the treadmill's speed, in order to keep the robot at a fixed distance from the front of the treadmill. This way, the robot can walk indefinitely and at various speeds, as long as it keeps walking

Tab. 4.2: The respective optimized parameters for the different terrains. You can see that the optimal step-length of the robot reduces over higher slopes, while the robot moves its hind feet more to the back. In (b), you can see that robot spends more time in the stance phase with higher slopes, and moves its hind feet further from his body.

grade	transfer learning	f	a	x_0	y_{0r}	y_{0f}	t_{stance}	l_{0f}	l_{1f}	l_{0r}	l_{1r}	l_2	l_3	speed (m/s)	Euclidean distance
flat	✗	1.98	95.3	137.0	-16.39	0.00	0.54	67.1	10.4	4.39	65.3	58.8	24.1	0.76	0
9%	✗	2.00	87.9	141.8	-11.88	0.00	0.60	70.0	3.13	0.00	47.6	69.4	4.53	0.67	4.13
9%	✓	2.00	90.6	140.1	-11.71	0.00	0.54	65.7	4.06	23.9	70.0	70.0	26.2	0.74	1.72
18%	✗	1.71	84.4	143.9	-8.87	0.00	0.56	52.6	34.8	0.00	51.6	66.2	4.51	0.53	6.64
18%	✓	2.00	86.3	142.8	-9.77	0.00	0.58	70.0	3.77	50.3	39.9	70.0	0.00	0.60	9.05

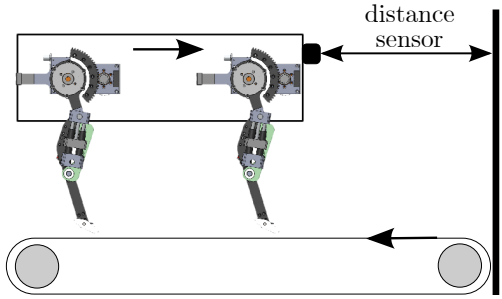


Fig. 4.4: Schematic representations of the setup, showing the robot on the treadmill. The robot is equipped with a distance sensor to detect the wall at the end of the treadmill.

in the forward direction. We test each gait for 4 seconds at a time, and have a smooth transition between these gaits for 2 seconds. Walking for more than an hour at a time poses little problem this way. In order to stop the robot from walking off the sides of the treadmill, a light thread has been added between the robot's head and an overhead rail, which limits the robot's lateral freedom of movement, while allowing it to freely move forwards. This thread is loose during normal operation. During the experiments, an assistant sits next to the track in order to intervene when the setup's safety is jeopardized by a very unstable gait.

4.6 Experimental setup

The goal of our experiments is to test our hypothesis: that transfer learning speeds up the learning of locomotion in compliant, quadrupedal robots. To carry this out, we compare the performance of an optimization with transfer learning, to the performance of an optimization without transfer learning. This way we can evaluate the benefit of transfer learning by comparing to a baseline in an identical setting. We verified this on an experimental basis by evaluating learning speed in three different classes of problems often encountered in robotics:

1. have a change in the environment of the robot, consisting of a different inclination of the treadmill,
2. have a change of the front leg's stiffness, i.e. a change in the robot's morphology,
3. increase the noise in the environment of the robot, by having the robot walk over pebbly terrain.

The goal of our optimization is to maximize the average robot speed, measured over a period of four seconds. In this paper, we chose to only optimize for speed. Initial experiments showed that additionally minimizing body rotations did not have any effect on the optimization process. Since adding this stability measure to the fitness would add a meta-parameter weighing the contribution of this measure, while hav-

ing no effect on the overall process, we chose to completely overhaul the stability as fitness and to only optimize for speed.

To evaluate these optimizations, we take a look at a number of metrics to evaluate the effectiveness of the optimization. Firstly, we compare the speed of the best gait in each optimization. This method compares the goals of the optimizations and is therefore necessary in the evaluation of the approaches. It is however limited, since it is very sensitive to outliers. This is especially a problem in our application, since due to our limited population size, we will comparatively have more outliers during the optimization process then when we would have used a larger population size.

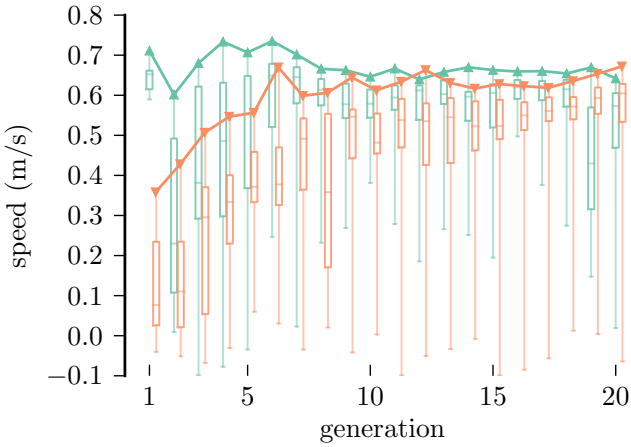
Therefore we also introduce a second metric, to evaluate whether all particles perform better on average than the particles of the same generation in the other optimization. To do this, we compare the speed of the gaits in each generation using a Wilcoxon rank-sum test, the non-parametric variant of the Student t-test, which is to be used for unknown distributions and small sample sizes [54]. Subsequently we combine the obtained p-values using Fisher's method, a well established method for doing so [55]. This way, we can test the significance of the claim that one optimization process is on average outperforming the other. This second method has the advantage of being less sensitive to outliers, as it uses more data and is a non-parameteric method. Note that we cannot say something on the statistical significance of the experiment itself, only on the significance of the difference between the two optimization processes compared to a population of speeds randomly drawn from a population.

The data from these experiments is laid out in the following sections, and serves not only to scrutinize our hypothesis, but also to explain the mechanisms behind them.

4.6.1 Transfer learning for different inclinations

In the first experiment, we evaluate the effectiveness of transferring a gait from a flat terrain to a slope. We start by optimizing a gait on a flat terrain using PSO. After that we optimize a new gait on a slope twice, once with transfer learning from the gaits on a flat terrain, once without transfer learning. We compare the obtained results with

(a) Optimizing on a 9% slope



(b) Optimizing on an 18% slope

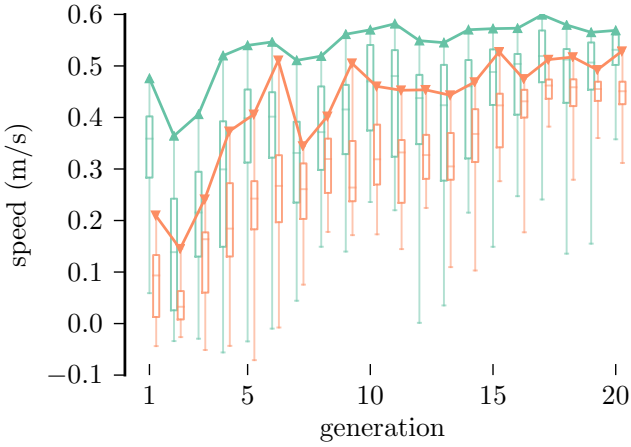


Fig. 4.5: Visualization of the particle scores for each generation: in green (▲) with transfer learning, in orange (▼) without. (a) the evolution of the fitness over a 9% slope. (b) evolution of the fitness for an 18% slope. Note the general better performance when transfer learning is used, especially when the slope is steeper and the problem is more difficult.

each other in order to evaluate the effect of transfer learning. We perform these last two optimizations on two inclinations (a 9% and an 18% grade, or a 5.14° and a 10.2° slope), making up a total of four optimizations.

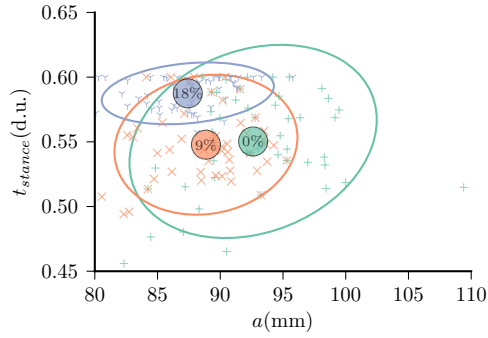
When we compare the achieved speeds of locomotion in Figure 4.5, it is clear that on both inclinations, the parameters not only perform better during optimization when using transfer learning, but they also train faster. We compare the results of the particles per generation using a Wilcoxon rank-sum test. We find that the particles in the optimization with transfer learning on average outperform the particles without transfer learning significantly on both inclinations ($\rho < 0.001$).

As we observe in Table 5.1, each inclination requires a different gait for optimal locomotion. We notice that on steeper inclinations, the step-length is reduced while the robot spends more time with its feet on the ground. We observe as well that the robot moves its hind feet further from the body and to the back, in order to level the body more and keep the center of mass within the support polygon.

When studying the best particles found during all of our optimizations, we found that there is a relation between the parameters found and the inclination on which the robot is running. As you can see in Figure 4.6, a higher inclination implies a lower variance in parameters. Thus trotting on a steeper terrain requires more specific parameters. Additionally, we notice a correlation between the parameters on the different inclinations. This confirms our previous research on computer models of a quadrupedal robot [56]. The results depicted in this figure indicate that a hierarchical learning approach to gaiting, such as the one used in [57], should prove fruitful. This is because the relation between the parameters can be learned, and this knowledge can subsequently speed up the learning process on intermediate or even higher inclinations. However, such an approach would lie outside the scope of this article.

In order to find out whether the method with transfer learning stays closer to the original particle, we have calculated the distances from the best particle on the flat terrain. In Table 5.1, we have included the Euclidean distance to the particle on the flat terrain, after normalization of the parameters. This shows that the result of the optimization

(a) The step length a and the fraction of time in the stance phase t_{stance} of the 50 best particles at each incline grade.



(b) The step height x_0 and a spline parameter l_{1f} of the 50 best particles in each incline grade.

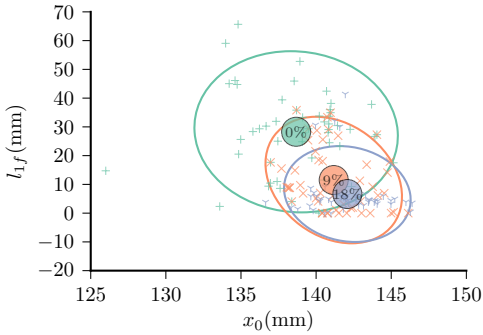


Fig. 4.6: In these figures, we have plotted the parameters of the 50 best particles found in all our optimizations for each inclination. We have fitted a Gaussian distribution to these particles, and depicted the mean together with the two standard deviations ellipse. We see that the highest variance is found on the flat terrain, lowering with an increasing inclination. This indicates that walking on higher inclinations has a higher parameter sensitivity, and is thus more difficult to optimize. Secondly, we see that there is a relation between the parameters and the inclinations. With increasing slope, the best gaits have a longer stance phase, a shorter step length and the feet move further from the body.

with transfer learning is not necessarily closer to the original solution, which serves as an indication that enough exploration does happen. Apparently the distance from the optimal parameters on a flat terrain increases with increasing inclination as well. Looking at the speed of the optimized results, we can see that the best particles from transfer learning have a gait with a speed 9% and 14% faster on the 9% and 18% grades respectivey.

These results confirm our hypotheses, namely that transfer learning increases the speed of the optimization process, resulting in better particles throughout the optimization process.

Tab. 4.3: The respective optimized parameters for the different stiffness of the front leg springs.

	transfer learning	f	a	x_0	y_{0r}	y_{0f}	t_{stance}	l_{0f}	l_{1f}	l_{0r}	l_{1r}	l_2	l_3	speed (m/s)	Euclidean distance
normal	✗	1.98	95.3	137.0	-16.39	0.00	0.54	67.1	10.4	4.39	65.3	58.8	24.1	0.76	0
compliant	✗	1.73	96.2	136.4	-24.16	0.00	0.50	0.00	1.74	0.00	40.7	70.0	25.2	0.66	12.98
compliant	✓	2.00	105.7	129.1	-5.27	0.00	0.60	70.0	0.00	0.00	70.0	70.0	70.0	0.75	7.16

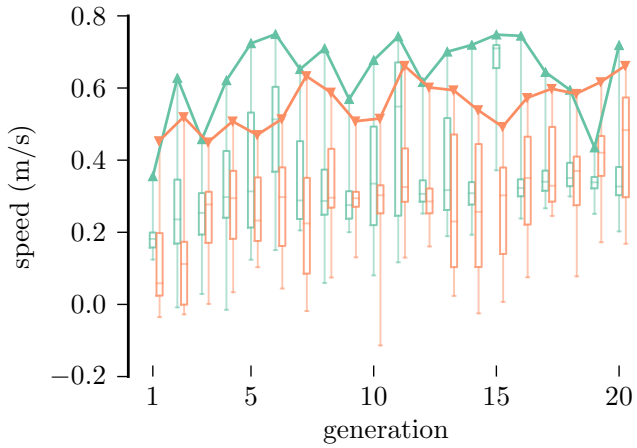


Fig. 4.7: Visualization of the particle scores for each generation when optimizing for a reduced stiffness. In green (▵) with transfer learning, in orange (▿) without. Notice that despite the original particles perform worse, the overall optimization is still better. This means that the better starting position of the transferred particles are not the main reason why they perform better.

4.6.2 Transfer learning for different leg spring constants

In the second experiment, we have tested the effectiveness of transfer learning against changes in the robot's body parameters. To do so, we have reduced the spring constant in the front legs of the robot by removing one of the two springs in each of the robot's front legs. These springs can be seen on Figure 4.1b. This reduction of the front leg stiffness results in a decrease of their pushing capacity. We use the same procedure as we did in the previous experiment with different inclinations, comparing the optimization results with and without transfer learning.

If we look at the data in Figure 4.7, we observe that the original transferred particles do not perform as well as the random particles on the new task. However, the improvement of the particles in the first generations is still higher than the one of the particles without transfer learning. This is remarkable, because in order to perform better, the transferred particles not only have to learn faster than their counterparts, but they also have to overcome the disadvantage of starting with a worse gait. We will offer an explanation for this observation in a dedicated subsection further in the paper.

Using the same statistical method as before, we find that the particles in the optimization process with transfer learning run significantly larger distances ($\rho < 0.05$).

If we compare the best parameters, as shown in Table 4.3, we can see that the best particle of the optimization with transfer learning has a 13% faster gait than in the optimization process without transfer learning. This is despite the fact that the solution is quite different from the solutions with the normal stiffness which were transferred originally, as can be seen in Table 4.3. The Euclidean distance of the optimized solution to the particle with the normal stiffness is relatively large, both with and without transfer learning. We will offer an explanation for this behaviour in the section 'The Mechanism Behind the Speedup'.

Once more, these results confirm our claim, that transfer learning increases the speed of the optimization process. Equivalently, the gaits perform better after a given time of optimizing. Moreover, this experiment shows that in transfer learning, there is another mechanism

at work besides having a head start.

4.6.3 Transfer learning for difficult terrains

As a third experiment, to further scrutinize our hypothesis, we have optimized in a noisier environment. In noisy problems, the PSO will produce more robust parameters in order to find gaits that perform well, because the same parameters do not always obtain the exact same result. An ill placed pebble could potentially tip over the robot, while there might be no problem if the pebbles are arranged slightly differently. In order to do this, we had the robot walking over a flat surface covered in pebbles. We attached a pebble dispenser to our treadmill, which covered the treadmill in pebbles with a diameter of approximately 1 cm. A video of this setup is available online[1].

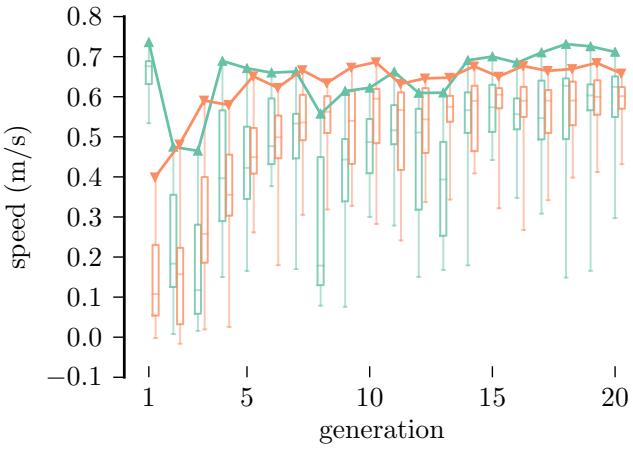


Fig. 4.8: In this figure we visualize the scores of the particles for each generation when optimizing for a more difficult terrain. In green (▲) with transfer learning, in orange (▽) without. Notice that despite all the original particles outperform the random particles in the first generation, this head start is quickly lost during the rest of the optimization process.

If we look at the results in Figure 4.8, we see that the original transferred particles perform well on this new task. However, once the

[1] http://youtu.be/kcBBdwwYmQA

PSO-algorithm starts exploring, it is hard to return to these original good solutions. If we compare both optimization processes using the same method as before, we find that we cannot confirm our hypothesis. Over all generations together, the particles optimized with transfer learning do not perform significantly better than the particles without transfer learning ($\rho > 0.05$).

If we look at the best particles in Table 4.4, we see that the one from the transfer learned optimization does not differ a lot from the original optimization on the flat terrain. The Euclidean distance to the original particle is comparatively small. This is because the best particle found during the entire optimization was one of the transferred particles. If we look at the best particle in the same optimization, but excluding the transferred particles, we find that it is very similar to these solutions as well.

Tab. 4.4: The respective optimized parameters on rocky and normal terrain, with and without transfer learning. We have also added the parameters of the second best particle in the transfer learning case, because the best particle was part of the first generation

terrain	transfer learning	f	a	x_0	y_{0r}	y_{0f}	t_{stance}	l_{0f}	l_{1f}	l_{0r}	l_{1r}	l_2	l_3	speed (m/s)	Euclidean distance
normal	✗	1.98	95.3	137.0	-16.39	0.00	0.54	67.1	10.4	4.39	65.3	58.8	24.1	0.76	0
pebbly	✗	2.00	87.3	142.2	3.44	0.00	0.49	0.00	70.0	60.6	70.0	70.0	70.0	0.69	25.82
pebbly	✓	2.00	89.3	141.0	-18.48	0.00	0.59	54.6	34.9	0.08	64.9	55.2	22.0	0.74	2.44
pebbly	✓	2.00	94.8	137.3	-8.29	0.00	0.57	70.0	29.0	0.00	63.2	56.4	28.7	0.74	1.25

4.7 The mechanism behind the speedup

The previous results show that the optimization time decreases when using particles from similar but different problems in the initialization step. Our experiment with the changed stiffness indicates that this is not necessarily caused by the transferred particles scoring well in the new problem. The experiment on the pebbly terrain indicates the same thing. Even though these particles started off better, they did not outperform their counterparts without transfer learning. In this section we show an alternative mechanism behind the transfer learning, namely that the learning process is speeded up because the

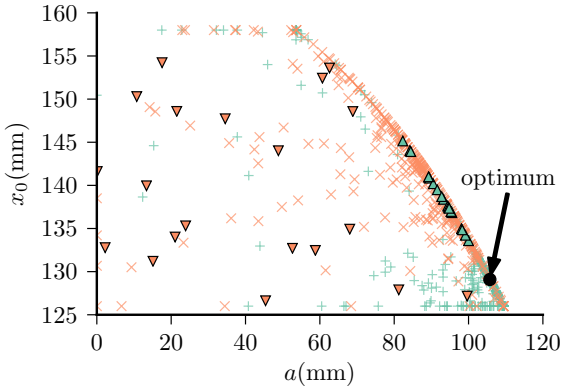
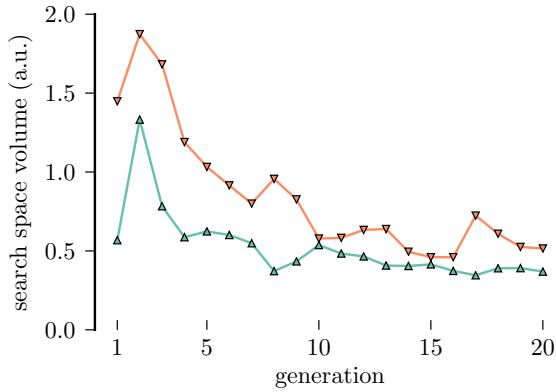


Fig. 4.9: A chart showing the values of the step length a and the distance from the body x_0 evaluated during the PSO. The parameters of two optimization processes are shown: the green particles (+, ▲) have been optimized with transfer learning, the orange particles (×, ▼) without. The triangles (▲, ▼) show the parameters at the initialization of the PSO. The rightmost edge of the search space is not vertical, because a_{max} depends on x_0 as explained by Table 4.1. Notice how the transferred parameters reduce the search space for the optimization algorithm, whereas the randomly initialized parameters still need to explore the entire parameter space before finding that maximizing the step length results in faster gaits. Note that most particles lie on the edge of the parameter range, which is also the end of the robot's range of motion. When the particles have been transferred, there is still a little exploration around the entire parameter space, but most particles are centered around the optimum.

(a) Optimizing on a 9% slope



(b) Optimizing on an 18% slope

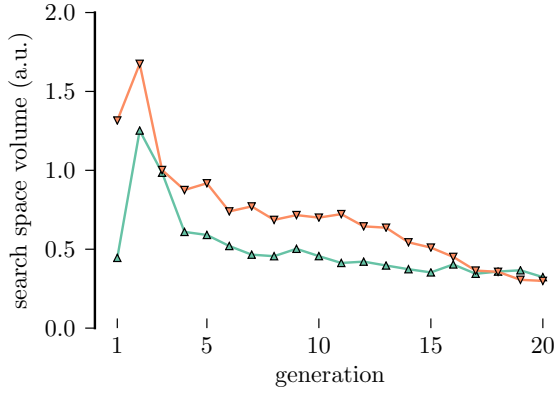
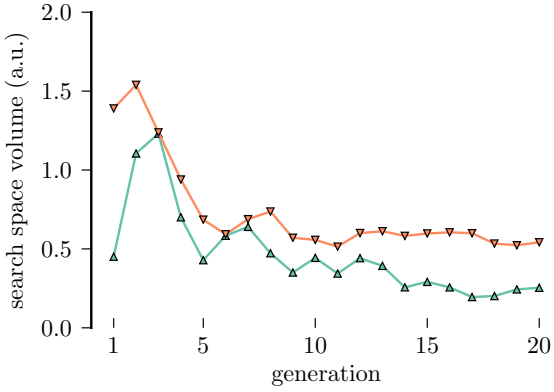
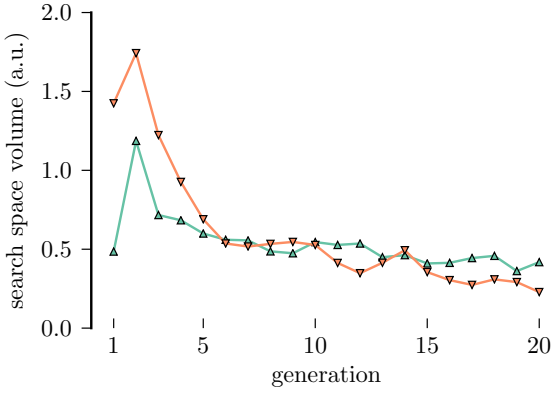


Fig. 4.10: The size of the search space is plotted here as the geometric mean of the lengths of the principle component axis of the particle parameters for each generation. In green (▲) with transfer learning, in orange (▼) without. The lower this mean, the smaller the volume of the parameter space is which is being explored in that generation. As you can see, the transferred particles initially occupy a smaller part of the parameter space, and this head start is not often lost during the optimization process. This is an important second mechanism through which is the optimization speed is increased.

(a) Optimizing with reduced stiffness of the front legs



(b) Optimizing on a rocky terrain



The size of the search space is plotted here as the geometric mean of the lengths of the principle component axis of the particle parameters for each generation. In green (▲) with transfer learning, in orange (▼) without. The lower this mean, the smaller the volume of the parameter space is which is being explored in that generation. As you can see, the transferred particles initially occupy a smaller part of the parameter space, and this head start is not often lost during the optimization process. This is an important second mechanism through which is the optimization speed is increased.

transferred particles indicate useful areas of the parameter space to explore.

To explain this, we plot all tested parameter combinations of the optimization on the 18% grade inclination. As you can see in Figure 4.9, it is more fruitful to maximize the step length a and have the feet moving at a medium distance from the body. You can see that in the transferred parameters, this general idea is already contained in the original particles, while the random parameters still need to discover this relation. So even though the transferred particles do not perform well, they already contain this general idea. Therefore, the search space is reduced. This way, the optimization process has to spend less time on rediscovering this relationship, and can focus on the more fruitful areas of the parameter space.

To make this clearer, we have plotted in Figure 4.10 the mean size of the principal component axes of the particles throughout the optimization process. These axes are an indicator of the size of the part of the parameter space that is being explored in a particular generation. The size of search space with transfer learning is on average 1.7 times smaller. Moreover, the transferred parameters start with a search space which is about the same size as at the end of the optimization without transfer learning. This indicates again that the increased learning speed when optimizing with transferred particles can be explained by the reduced search space in which the search starts.

4.8 Conclusion

In this article, we have found that transfer learning is beneficial for learning gaits on our legged robot. In order to test our hypothesis that transferring particles from previous optimizations improves the speed of learning new gaits on different problems, we compared these optimizations with optimizations starting from random samples. We optimized a gait for our robot in three different settings: first on a 9% and an 18% grade inclination, then with a different leg stiffness and finally on a pebbly terrain. We found that using transfer learning results in better gaits than without transfer learning in all tested cases, evaluated on a real robot. Everything considered, we conclude

that transferring particles resulted in better gaits in the same amount of robot time in three out of four cases, while not harming the optimization speed in the fourth case.

We have also shown that the increased learning speed is caused by a reduction of the volume in the parameter space that is used in the exploration. Therefore, we believe that our approach is more generally applicable and will serve useful for further developments in walking robots.

How generally applicable this conclusion is, cannot be reliably determined from this study alone, since only a single robot and a limited set of walking conditions was used to obtain the data. We did however identify a mechanism, alter the robot parameters and the robot surroundings to test the robustness of this method and found similarities between the results of the different optimizations, which gives us confidence for a more general applicability.

The research reported here indicates that this idea of transferring particles is more broadly applicable in particle swarm optimization. In general, it would be interesting to verify whether the mechanism of the reduced search space is also the main mechanism when applying transfer learning in other optimization algorithms. Especially because of this mechanism, we believe that this approach might be beneficial for other learning problems in robotics as well, where the number of evaluations in an optimization is inherently limited and tasks are often similar, but solutions to one problem will not often work well on another.

Acknowledgements

We would like to thank Benjamin Schrauwen for the fruitful initial discussions and his practical advice during the experiments. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 – Challenge 2 – Cognitive Systems, Interaction, Robotics – under grant agreement No 248311 – AMARSi, and from the Agency for Innovation by Science and Technology in Flanders (IWT).

Bibliography

[1] J. Degrave, M. Burm, P.-J. Kindermans, J. Dambre, and F. wyffels, "Transfer learning of gaits on a quadrupedal robot," *Adaptive Behavior*, vol. 23, no. 2, pp. 69–82, 2015.

[2] G. A. Pratt, "Legged robots at MIT: what's new since raibert?" *Robotics & Automation Magazine, IEEE*, vol. 7, no. 3, pp. 15–19, 2000.

[3] B. Verrelst, R. Van Ham, B. Vanderborght, F. Daerden, D. Lefeber, and J. Vermeulen, "The pneumatic biped "Lucy" actuated with pleated pneumatic artificial muscles," *Autonomous Robots*, vol. 18, no. 2, pp. 201–213, 2005.

[4] M. Raibert, K. Blankespoor, G. Nelson, R. Playter *et al.*, "Bigdog, the rough-terrain quadruped robot," in *Proceedings of the 17th World Congress*, 2008, pp. 10 823–10 825.

[5] N. G. Tsagarakis, Z. Li, J. Saglia, and D. G. Caldwell, "The design of the lower body of the compliant humanoid robot "cCub"," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 2035–2040.

[6] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schäffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald, and G. Hirzinger, "The KUKA-DLR lightweight robot arm-a new reference platform for robotics research and manufacturing," in *41st international symposium on Robotics (ISR) and 6th German conference on robotics (ROBOTIK)*. VDE, 2010, pp. 1–8.

[7] J. Pratt and B. Krupp, "Design of a bipedal walking robot," in *SPIE Defense and Security Symposium*. International Society for Optics and Photonics, 2008, pp. 69 621F–69 621F.

[8] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D. G. Caldwell, "Design of hyq–a hydraulically and electrically actuated quadruped robot," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, p. 0959651811402275, 2011.

[9] M. Hutter, C. Gehring, M. Bloesch, M. Hoepflinger, C. D. Remy, and R. Siegwart, "StarlETH: A compliant quadrupedal robot

for fast, efficient, and versatile locomotion," in *15th International Conference on Climbing and Walking Robot (CLAWAR)*, no. EPFL-CONF-181042, 2012.

[10] B. K. Horn and M. H. Raibert, "Configuration space control," 1977.

[11] R. Dillmann, O. Rogalla, M. Ehrenmann, R. Zollner, and M. Bordegoni, "Learning robot behaviour and skills based on human demonstration and advice: the machine learning paradigm," in *International Symposium on Robotics Research (ISRR)*, vol. 9, 2000, pp. 229–238.

[12] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3. IEEE, 2004, pp. 2619–2624.

[13] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun, "Learning motion patterns of people for compliant robot motion," *The International Journal of Robotics Research (IJRR)*, vol. 24, no. 1, pp. 31–48, 2005.

[14] J. R. Peters, "Machine learning of motor skills for robotics," Ph.D. dissertation, University of Southern California, 2007.

[15] E. T. Wolbrecht, V. Chan, D. J. Reinkensmeyer, and J. E. Bobrow, "Optimizing compliant, model-based robotic assistance to promote neurorehabilitation," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 16, no. 3, pp. 286–297, 2008.

[16] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011, pp. 465–472.

[17] T. Waegeman, F. wyffels, and B. Schrauwen, "Feedback control by online learning an inverse model," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 10, pp. 1637–1648, 2012.

[18] M. E. Taylor, S. Whiteson, and P. Stone, "Transfer via inter-task mappings in policy search reinforcement learning," in *Proceedings*

of the 6th international joint conference on Autonomous agents and multiagent systems. ACM, 2007, p. 37.

[19] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *The Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.

[20] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[21] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: transfer learning from unlabeled data," in *Proceedings of the 24th international conference on Machine learning.* ACM, 2007, pp. 759–766.

[22] C. Do and A. Ng, "Transfer learning for text classification," in *NIPS*, 2005.

[23] S. Thrun and T. M. Mitchell, *Lifelong robot learning.* Springer, 1995.

[24] M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich, "To transfer or not to transfer," in *NIPS 2005 Workshop on Transfer Learning*, vol. 898, 2005.

[25] J. R. Rebula, P. D. Neuhaus, B. V. Bonnlander, M. J. Johnson, and J. E. Pratt, "A controller for the littledog quadruped walking on rough terrain," in *IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 2007, pp. 1467–1473.

[26] K. Byl, A. Shkolnik, S. Prentice, N. Roy, and R. Tedrake, "Reliable dynamic motions for a stiff quadruped," in *Experimental robotics.* Springer, 2009, pp. 319–328.

[27] F. Iida and R. Pfeifer, "Cheap rapid locomotion of a quadruped robot: Self-stabilization of bounding gait," in *Intelligent Autonomous Systems*, vol. 8, 2004, pp. 642–649.

[28] C. Niehaus, T. Röfer, and T. Laue, "Gait optimization on a humanoid robot using particle swarm optimization," in *Proceedings of the Second Workshop on Humanoid Soccer Robots (IEEE-RAS)*, 2007.

[29] F. wyffels, M. D'Haene, T. Waegeman, K. Caluwaerts, C. Nunes, and B. Schrauwen, "Realization of a passive compliant robot dog," in *3rd IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. IEEE, 2010, pp. 882–886.

[30] J. Degrave, M. Burm, T. Waegeman, F. wyffels, and B. Schrauwen, "Comparing trotting and turning strategies on the quadrupedal oncilla robot," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2013.

[31] M. R. Dimitrijevic, Y. Gerasimenko, and M. M. Pinter, "Evidence for a spinal central pattern generator in humansa," *Annals of the New York Academy of Sciences*, vol. 860, no. 1, pp. 360–376, 1998.

[32] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: a review," *Neural Networks*, vol. 21, no. 4, pp. 642–653, 2008.

[33] Y. Sakakibara, K. Kan, Y. Hosoda, M. Hattori, and M. Fujie, "Foot trajectory for a quadruped walking machine," in *IEEE International Workshop on Intelligent Robots and Systems. 'Towards a New Frontier of Applications', IROS Proceedings.* IEEE, 1990, pp. 315–322.

[34] C. Maufroy, H. Kimura, and K. Takase, "Integration of posture and rhythmic motion controls in quadrupedal dynamic walking using phase modulations based on leg loading/unloading," *Autonomous Robots*, vol. 28, no. 3, pp. 331–353, 2010.

[35] V. Barasuol, J. Buchli, C. Semini, M. Frigerio, E. R. De Pieri, and D. G. Caldwell, "A reactive controller framework for quadrupedal locomotion on challenging terrain," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 2554–2561.

[36] L. Shen and R. E. Poppele, "Kinematic analysis of cat hindlimb stepping," *Journal of neurophysiology*, vol. 74, no. 6, pp. 2266–2280, 1995.

[37] F. Flower, D. Sanderson, and D. Weary, "Hoof pathologies influence kinematic measures of dairy cow gait," *Journal of dairy science*, vol. 88, no. 9, pp. 3166–3173, 2005.

[38] H. Kimura, I. Shimoyama, and H. Miura, "Dynamics in the dynamic walk of a quadruped robot," *Advanced Robotics*, vol. 4, no. 3, pp. 283–301, 1989.

[39] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[40] J. Kennedy, "The particle swarm: social adaptation of knowledge," in *IEEE International Conference on Evolutionary Computation*. IEEE, 1997, pp. 303–308.

[41] R. Poli, "An analysis of publications on particle swarm optimization applications," *Essex, UK: Department of Computer Science, University of Essex*, 2007.

[42] Y. Qin, D. Sun, N. Li, and Q. Ma, "Path planning for mobile robot based on particle swarm optimization," *Robot*, vol. 26, no. 3, pp. 222–225, 2004.

[43] A. Chatterjee, K. Pulasinghe, K. Watanabe, and K. Izumi, "A particle-swarm-optimized fuzzy-neural network for voice-controlled robot systems," *IEEE Transactions on Industrial Electronics*, vol. 52, no. 6, pp. 1478–1489, 2005.

[44] J. Pugh, A. Martinoli, and Y. Zhang, "Particle swarm optimization for unsupervised robotic learning," in *Proceedings of IEEE swarm intelligence symposium (SIS)*. Citeseer, 2005, pp. 92–99.

[45] J. Pugh and A. Martinoli, "Distributed adaptation in multi-robot search using particle swarm optimization," in *From Animals to Animats 10*. Springer, 2008, pp. 393–402.

[46] T. Hemker, M. Stelzer, O. von Stryk, and H. Sakamoto, "Efficient walking speed optimization of a humanoid robot," *The International Journal of Robotics Research*, vol. 28, no. 2, pp. 303–314, 2009.

[47] N. Shafii, S. Aslani, O. M. Nezami, and S. Shiry, "Evolution of biped walking using truncated fourier series and particle swarm optimization," in *RoboCup 2009: Robot Soccer World Cup XIII*. Springer, 2010, pp. 344–354.

[48] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE international conference on neural networks*, vol. 4, no. 2. Perth, Australia, 1995, pp. 1942–1948.

[49] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE, 1996, pp. 312–317.

[50] L. Davis, *Handbook of genetic algorithms*. Van Nostrand Reinhold New York, 1991, vol. 115.

[51] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *The 1998 IEEE International Conference on Evolutionary Computation Proceedings*. IEEE, 1998, pp. 69–73.

[52] M. E. H. Pedersen, "Good parameters for particle swarm optimization," *Hvass Lab., Copenhagen, Denmark, Tech. Rep. HL1001*, 2010.

[53] A. Sproewitz, L. Kuechler, A. Tuleu, M. Ajallooeian, M. D'Haene, R. Moeckel, and A. J. Ijspeert, "Oncilla robot: a light-weight bioinspired quadruped robot for fast locomotion in rough terrain," in *Procedures of the Fifth International Symposium on Adaptive Motion on Animals and Machines*, 2011.

[54] P. D. Bridge and S. S. Sawilowsky, "Increasing physicians' awareness of the impact of statistics on research outcomes: comparative power of the t-test and wilcoxon rank-sum test in small samples applied research," *Journal of clinical epidemiology*, vol. 52, no. 3, pp. 229–235, 1999.

[55] R. Rosenthal, "Combining results of independent studies." *Psychological bulletin*, vol. 85, no. 1, p. 185, 1978.

[56] P.-J. Kindermans, F. wyffels, K. Caluwaerts, B. Guns, and B. Schrauwen, "Towards incorporation of hierarchical bayesian models into evolution strategies for quadruped gait generation," in *21st Annual Belgian-Dutch conference on Machine Learning (BeNeLearn & PMLS)*. University Press, 2012, pp. 70–70.

[57] C. Kemp, A. Perfors, and J. B. Tenenbaum, "Learning overhypotheses with hierarchical bayesian models," *Developmental science*, vol. 10, no. 3, pp. 307–321, 2007.

5
Terrain Classification for a Quadruped Robot

J. Degrave, R. Van Cauwenbergh, F. wyffels, T. Waegeman, and B. Schrauwen, "Terrain classification for a quadruped robot," in *International conference on machine learning and applications (ICMLA)*. IEEE, 2013, pp. 185–190

⋆⋆⋆

Using data retrieved from the Puppy II robot at the University of Zurich (UZH), we show that machine learning techniques with non-linearities and fading memory are effective for terrain classification, both supervised and unsupervised, even with a limited selection of input sensors. We find that the classification error is small enough to have a robot adapt the gait to the terrain and hence move more robustly. The results indicate that most information for terrain classification is found in the combination of tactile sensors and proprioceptive joint angle sensors. Secondly, the results indicate the possible power of embodiment and morphological computation. Despite not having a dedicated sensor for classifying the terrain, we found that the difference in behavior of the limbs computed the features needed to classify the terrain already. Thirdly, the results indicate the trade-off between non-linearities and fading memory, a trade-off further explored in chapter 6.

5.1 Introduction

Terrain classification plays an important role in the control of legged robots, as it allows the robots to adapt to the terrain. On different terrains, different gaits will be more suitable and therefore a robot capable of switching from one gait to another in reaction to a terrain change, will be able to locomote more robustly.

There has been done some research on terrain classification for robots through sensor data from advanced sensors, such as camera imagery [2] or laser scanners [3]. This way, visual features in the terrain are used to discover the terrain type and subsequently the terrain properties.

Previous research also demonstrated the importance of proprioceptive sensors for amphibian robots, such as inertia-sensors, angle-encoders and current measurements on the motors [4]. Furthermore has it been shown for quadruped robots that force-sensing in the legs and current-use in the motors deliver reasonable results upon processing with an Adaboost algorithm [5]. Even only using proprioceptive and contact sensors proved effective in ground discrimination [6].

Firstly, perceiving and understanding the environment in which the robot operates has a high impact on the performance of the robot's locomotion, making it important to add sensors to the robot that provide information on the terrain. However, it is unfavorable to add unnecessary or complex sensors to the robot if they yield no further information. Those superfluous sensors would only increase the complexity of the robot design, while offering little possibilities for better control. Hence, it is important to know which sensors actually provide the most valuable data for terrain classification.

Secondly, instead of developing a single system which takes in raw data to directly determine the robot's actions, it can be better to divide the problem and conquer the easier sub-problems. One of the sub-problems is for the robot to recognize in real time the type of terrain while walking on it, using the data it receives from as few sensors as possible. Previous research in this area includes for instance the application of clustering techniques to detect transitions from one terrain to the next in order to achieve unsupervised classification,

applied on the RHex-robot [7, 8].

Therefore the research in this paper is twofold. Firstly, we want to identify which sensors provide most information on the terrain. In order to achieve this, we try different combinations of sensors often found in robots and evaluate the capability of supervised and unsupervised machine learning techniques to derive information from recorded data of those sensors. Secondly, we will also evaluate which machine learning techniques work best to classify the terrain based on these sensors, and which features are necessary for the techniques to function.

The rest of the paper is structured as follows. In Section 5.2 we will review the machine learning techniques used in this paper and go through Linear Regression (LR), Extreme Learning Machines (ELM), Reservoir Computing (RC), Slow Feature Analysis (SFA) and Independent Component Analysis (ICA). In Section 5.3 we will present the hardware used to retrieve the data and the methods used to process this data. In section 5.4 we will describe our experiments and results. Finally, conclusions will be drawn in section 5.5.

5.2 Machine Learning Techniques for Terrain Classification

5.2.1 Linear Regression

Linear regression (LR) is a supervised approach to modeling the relationship between K input variables and L scalar output variables. The relation between the two is described as:

$$\mathbf{W_{out}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

$\mathbf{X}$ is a matrix where every row is a time-step with in the columns the different input variables and $\mathbf{y}$ is a matrix and has in every row the corresponding desired outputs. With the transformation matrix $\mathbf{W_{out}}$ we can now process new signals $\mathbf{X}'$:

$$\hat{\mathbf{y}} = \mathbf{X}'\mathbf{W_{out}}. \tag{5.1}$$

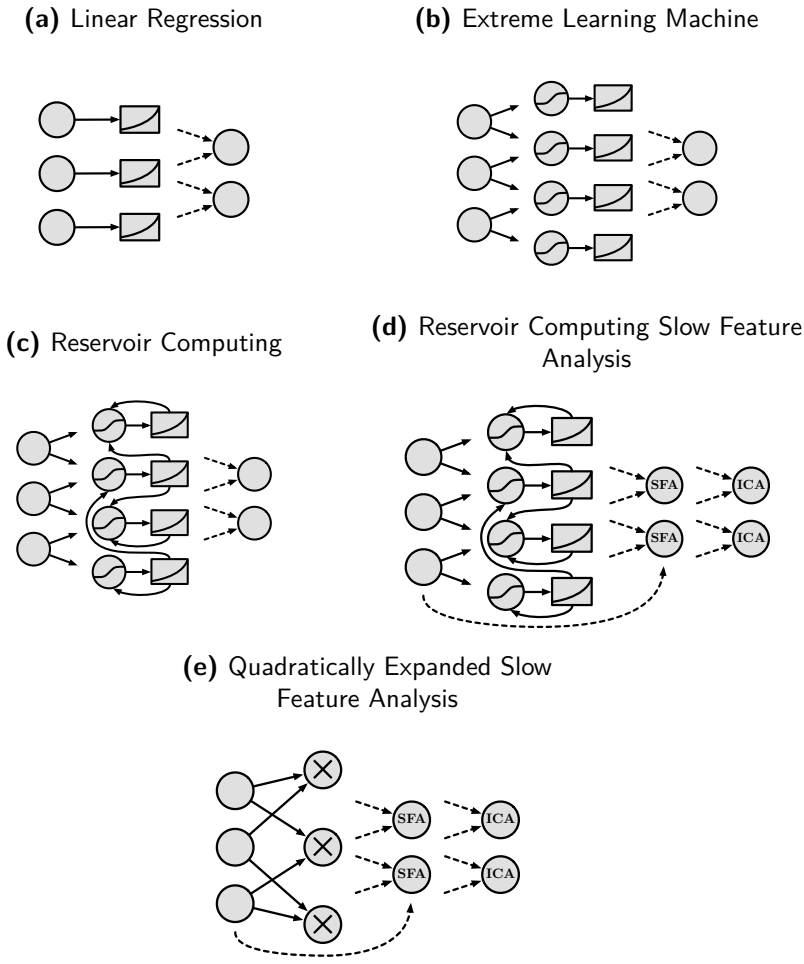


Fig. 5.1: Visual representation of the machine learning techniques used. The circles depict nodes, the rectangles filtering. Circles with a hyperbolic tangent curve have a non-linear activation; circles with a cross multiply inputs; rectangles with an exponential curve are low-pass filters. The solid arrows represent fixed weights, the dashed arrows represent trained weights.

The sensor signals are very noisy however, therefore we first low-pass filter these inputs with an exponential moving average (see Figure 5.1a), as shown in the equation below:

$$\mathbf{x}(n) = (1 - \alpha)\,\mathbf{x}(n-1) + \alpha\,\mathbf{u}(n). \tag{5.2}$$

Here, $\mathbf{x}(n)$ and $\mathbf{u}(n)$ are respectively the input of the linear regression and the sensor signal at time-step n. α is the leak rate. As the resulting signals still contains a lot of noise, no further regularization is needed.

5.2.2 Extreme Learning Machine

With the goal of having LR model the non-linearities more accurately, we first expand the sensor signals into a larger space by adding a hidden layer of non-linear nodes. The weights of these nodes are fixed, and are randomly selected from the set $\{-1, 0, 1\}$. This technique is similar to Extreme Learning Machine (ELM), a technique to train single-hidden layer feedforward neural networks [9]. The only difference from the standard implementation is that we use leaky nodes in the hidden layer to filter the noise in their state, as we did earlier with the linear regression. The update equations for these leaky ELMs are therefore given by:

$$\mathbf{x}(n) = (1 - \alpha)\,\mathbf{x}(n-1) + \alpha\,\tanh\left(\mathbf{W_{in}}\mathbf{u}(n)\right)$$

$$\mathbf{y}(n) = \mathbf{W_{out}}\mathbf{x}(n).$$

Here, $\mathbf{W_{in}}$ is an $N \times K$ matrix containing the fixed weights of the N nodes in the hidden layer. We model the non-linearity by using hyperbolic tangent nodes, as shown in Figure 5.1b. The matrix $\mathbf{W_{out}}$ is obtained by using LR, as discussed in section 5.2.1, but this time with the expanded set of signals as inputs.

5.2.3 Reservoir Computing

The ELM systems can be expanded further temporally and non-linearly by adding fixed weight connections between the nodes in the hidden layer, as shown in Figure 5.1c. The resulting system is

a Reservoir Computing system (RC). The term Reservoir Computing has been introduced in [10] to cover multiple previous computing techniques developed independently: Liquid State Machines (LSM) [11], Echo State Networks (ESN) [12] and BackPropagation DeCorrelation (BPDC) [13].

The weights of recurrent connections in the hidden layer nodes are fixed and randomly selected from a standard normal distribution. Similarly to the ESN in [14], the update equations of our reservoir computing systems are as follows:

$$\mathbf{x}(n) = (1-\alpha)\,\mathbf{x}(n-1) + \alpha\,\tanh\left(\mathbf{W_{res}}\mathbf{x}(n-1) + \mathbf{W_{in}}\mathbf{u}(n)\right)$$

$$\mathbf{y}(n) = \mathbf{W_{out}}\mathbf{x}(n).$$

$\mathbf{W_{res}}$ is an $N \times N$ matrix containing the fixed weights between the nodes in the hidden layer. After sampling the weights, the matrix is rescaled to have a spectral radius σ. Typically, σ is a good indicator of the echo state property [15] and is often chosen proximate to 1, close to the edge of chaos, where reservoir computing systems possess high computational power [16].

In Reservoir Computing, the hidden layer of recurrent nodes is often referred to as the reservoir. Because of the recurrent connections between the nodes, a fading memory is introduced into the system. We already inserted some memory with leaky nodes, but the hidden layer nodes in an ELM cannot act dynamically at a certain time-step based on the result of the previous time-step. Therefore, the type of memory used in LR and ELM does not add dynamics, but merely serves as a noise filter, opposed to the dynamic properties in a reservoir. Consequently, RC has memory capacity [17] as RC systems can learn relations with the past, while ELM has no memory capacity due to the lack of recurrent connections between the hidden-layer nodes.

5.2.4 Unsupervised learning

It would be interesting for an autonomous learning robot to have it learn terrains autonomously as well, without being shown a distinction between terrains in advance. To achieve this, we continue from

the non-linear, temporal expansion created by the reservoir, as the dynamics generated in a reservoir are suited for terrain classification, albeit supervised, which we will establish in section 5.4.1.2.

However, to make the system unsupervised, we process the output of the reservoir further with unsupervised techniques instead of using LR. First, we apply slow feature analysis (SFA) to derive the slow changing features in the output of the reservoir. Secondly, we apply independent component analysis (ICA) to these features in order to find the maximally statistically independent features. The complete setup is depicted in Figure 5.1d. This method has already been used in the context of robotics for robot localization [18]. There, they referred to the output of the ICA-layer as place cells, because they behave similarly to nodes found in the hippocampus of rodents [19]. Contrary to the place cells in rodents, which fire at a certain location, we have created *terrain* cells, which fire on a certain terrain.

As a baseline comparison for the terrain cells obtained by using reservoir computing and slow feature analysis (RC-SFA), we also expand the input signals quadratically instead of using a reservoir. This means that we use the product of each combination of 2 different input signals alongside the original signals as input for the slow feature analysis, shown in Figure 5.1e. We will refer to this system as SFA2.

5.2.4.1 Slow Feature Analysis

Slow Feature Analysis is an unsupervised machine learning algorithm which extracts slow varying signals from faster varying signals [20]. Suppose $\mathbf{x}(t)$ is a multi-dimensional input signal. SFA generates a slow varying output signal $y_i(t) = g_i(\mathbf{x}(t))$ by searching for the best functions g_i in a certain space such that

$$\Delta(y_i) = \langle \dot{y}_i^2 \rangle_t \tag{5.3}$$

is minimized under the following conditions:

$$\langle y_i \rangle_t = 0 \tag{5.4}$$

$$\langle y_i^2 \rangle_t = 1 \tag{5.5}$$

$$\forall j < i, \langle y_i y_j \rangle_t = 0. \tag{5.6}$$

This can be solved efficiently by the algorithm proposed in [20].

5.2.4.2 Independent Component Analysis

Independent Component Analysis [21] is an algorithm that separates multivariate signals in order to achieve maximally uncorrelated signals. It assumes there is a linear connection between the input signals $\mathbf{x}(t)$ and the underlying uncorrelated signals $\mathbf{s}(t)$. Therefore, the maximally uncorrelated signals can be regenerated using a matrix $\mathbf{W_{ICA}}$:

$$\mathbf{s}(t) = \mathbf{W_{ICA}}\mathbf{x}(t). \tag{5.7}$$

To find this matrix $\mathbf{W_{ICA}}$, we use the FastICA-algorithm [22].

5.2.5 Covariance Matrix Adaptation Evolution Strategy

To optimize the parameters of the different techniques described above, we use *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES). CMA-ES is an evolutionary optimization algorithm which makes very few assumptions on the nature of the system. Only the ranking between the candidates is used in the learning process, no gradients or quantification of the input is needed. CMA-ES proves to be useful for non-separable, ill-conditioned or noisy objective functions [23]. Also, the different evaluations within one generation are independent, so the algorithm can easily be parallelized. In this paper, we use the implementation of CMA-ES written by Nikolaus Hansen [23].

CMA-ES has only one free parameter left, the population size λ. For this parameter, we use $\lambda = \lfloor 4 + 3\ln(M) \rfloor$ with M the number of parameters which need to be optimized. This choice has been made as a balance between quick convergence and the amount of parallelization possible, but is still arbitrary. The λ-parameter functions on a meta-level and therefore has little effect on the conclusions made in this paper.

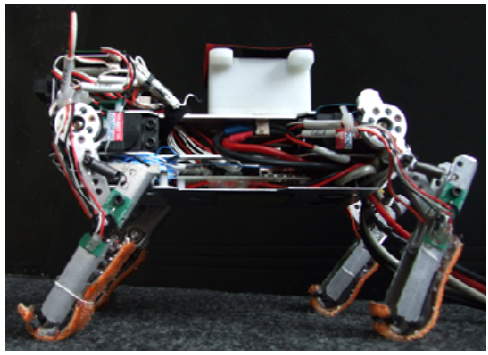


Fig. 5.2: Puppy II, from the Artificial Intelligence Laboratory, Department of Informatics, University of Zurich. This is a side-view with the front of the robot on the left.

5.3 Methodology

5.3.1 The Dataset

The dataset used in this work has been harvested at the University of Zurich, using their quadruped robot, Puppy II (Fig. 5.2) [24]. This robot has four identical legs, each controlled by a single servomotor at the hip joint. The knee joints are not directly controllable, but are passively moveable due to the spring attached between the upper and lower limb. Underneath the feet there is an adhesive skin attached with asymmetrical friction. This aids the robot in moving its legs forward and enforcing its grip while moving the legs backward. Puppy II is fitted with a number of sensors. There is a potentiometer attached to each joint, for a total of eight potentiometers. Each leg has a tactile sensor which measures the force exerted on the paw when touching the ground. The robot has an IMU and there is also a single external sensor used, namely an overhead camera. This camera is not used directly, but the position and velocity derived from the camera footage is.

The trials have a varying length, many between 3000 and 6500 samples with a sampling rate of 50Hz. The measurements have been made on five different terrains, at seven different gait frequencies, with three different gaits. The *bound right* and the *turn left* gait are generated by simple sine waves on the joints, with a different offset, amplitude and

Tab. 5.1: Number of available trials in function of terrain type, gait frequency and gait. (F: blue foil, S: Styrofoam, L: linoleum, C: Cardboard, R: rubber)

	bound right					turn left					random				
	F	S	L	C	R	F	S	L	C	R	F	S	L	C	R
0.25 Hz	3	1	-	-	-	2	1	-	-	-	-	-	-	-	-
0.50 Hz	5	2	-	-	-	4	2	-	-	-	-	-	-	-	-
0.75 Hz	4	2	-	-	-	4	2	-	-	-	-	-	-	-	-
1.00 Hz	2	13	9	3	5	4	4	3	-	-	4	3	4	-	-
1.25 Hz	6	1	-	-	-	-	3	-	-	-	-	-	-	-	-
1.52 Hz	3	2	-	-	-	-	6	-	-	-	-	-	-	-	-
1.72 Hz	8	3	-	-	-	-	-	-	-	-	-	-	-	-	-

phase for each motor. In the *bound right* gait, the robot moves in a clockwise circle due to a slightly higher amplitude on the left legs. In the *turn left* gait, the robot moves counter-clockwise due to a much larger amplitude on the right hind leg. The *random* gait consists of random motor commands, sufficiently smooth not to exceed the motor bandwidth. For an overview of the trials available for each combination of parameters, we refer to Table 5.1.

After testing a few combinations of sensor signals, we settle on a selection that delivers an optimal or near-optimal performance for all techniques that we examine. In this paper, we will mainly evaluate and compare the use of these eighteen signals: the angles of the four hips, the angles of the four knees, the tactile sensors on the four feet, the three-dimensional accelerometer and the three-dimensional gyroscope. For completeness' sake, we also evaluate the use of all information available in the dataset. We normalize each sensor signal to a zero mean and a unit variance. Then we select a total of 3000 samples for each terrain type and do this five times. Three of these are used for threefold cross-validation while optimizing the parameters, the other two are used for twofold cross-validation during the verification of the parameters. Any remaining relevant trials are added to the test set during verification. We split all trials into parts as a final step, with lengths varying from 281 to 562 samples. Most of these are

Tab. 5.2: The different groups of sensors and the sensor information they contain

	useful	reduced	minimal	leg	agt	imu	joints	touch
Number of dimensions	55	21	18	12	10	9	8	4
Tactile sensors	×	×	×	×	×			×
Joint angle sensors	×	×	×	×			×	
Accelerometer and gyroscope	×	×	×		×	×		
Velocity	×	×						
Compass	×	×						
Distance to wall	×	×						
Magnetometer	×					×		
Other/derivative sensors	×							

about 375 samples long (7.5s).

These sensors were combined into a couple of groups which we evaluate on how much information they contain on the terrain type. For a detailed description of what sensor group contains which sensors, we refer to Table 5.2.

5.3.2 System Classification Score

To evaluate the accuracy of the methods, the terrain is classified at each time-step. This means that we measure the effectiveness of the techniques as a realtime sensor. For the supervised techniques, we have 5 output nodes, the same as the number of terrains in the dataset. We train the output nodes to be 1 when the robot is walking on their respective terrain and to be -1 when the robot is not walking on its terrain. At each time-step, we classify by choosing the terrain corresponding to the output node with the highest value. The score of the system is the percentage of the time-steps correctly classified.

This approach however does not work for unsupervised techniques, as

they do not have corresponding output nodes, as the system did not have any example data. To be able to give meaning to the output of the ICA-layer, we reconstruct the probability that the robot is on a certain terrain, given the terrain cells, with Bayes' theorem:

$$P(\mathbf{x_r}|\mathbf{y_{ICA}}) = \frac{P(\mathbf{y_{ICA}}|\mathbf{x_r})P(\mathbf{x_r})}{P(\mathbf{y_{ICA}})} \tag{5.8}$$

where $P(\mathbf{x_r})$ is the prior on the terrain vector and is related to our dataset and $P(\mathbf{y_{ICA}})$ is a normalization factor which does not need to be calculated explicitly. The classifier picks the terrain with the highest probability of $P(\mathbf{x_r}|\mathbf{y_{ICA}})$ as the correct terrain. The score of the unsupervised system, is the percentage of time-steps correctly classified this way.

$P(\mathbf{y_{ICA}}|\mathbf{x_r})$ can be obtained as follows, since the outputs of the ICA layer are statistally independent:

$$P(\mathbf{y_{ICA}}|\mathbf{x_r}) = \prod_{i=1}^{N_{ICA}} P(y^i_{ICA}|\mathbf{x_r}) \tag{5.9}$$

with y^i_{ICA} the output of terrain cell i. Finally, we can estimate $P(y^i_{ICA}|\mathbf{x_r})$ based on our train set, by creating a histogram of the terrain cells given a certain terrain.

5.4 Experiments

5.4.1 Supervised

5.4.1.1 Sensor Selection

The first experiment determines which sensor combinations provide the best information for supervised terrain classification. We test the eight different sensor combinations ranging from 4 to 55 signals out of 64 available in the dataset.

We process the sensor data of these combinations with PCA to compress the input signals, with the dimension reduction rate as a parameter. The compressed signals are then classified with LR and RC.

Tab. 5.3: Train and test results for eight signal selections (in percentage, more is better). The entire table has a single color gradient: maximal value is green, minimal is red, average of the two is yellow.

	TRAIN		TEST	
	LR	RC	LR	RC
`useful`	89.06	95.16	71.53	76.30
`reduced`	79.66	96.27	64.08	79.36
`minimal`	82.28	95.96	67.36	81.62
`leg`	75.70	95.42	63.63	81.53
`agt`	73.36	95.68	66.69	82.84
`imu`	53.35	91.47	43.98	72.25
`joints`	59.85	91.99	47.52	74.12
`touch`	64.45	93.84	57.89	84.69

All parameters of the different techniques are optimized using CMA-ES for each combination of sensors. In Table 5.3 the percentage of correctly classified time-steps is shown.

Note that the dataset has few trials for each combination of terrain, gait and gait frequency. Only some of these combinations are included in the training set, which increases the difficulty of correctly classifying the entire test set as the algorithms need to generalize over gaits and frequencies. Consequently there is a gap between the performance on the train set and the performance on the test set. This demonstrates the correlation of terrain classification performance and the actions of the robot, confirming the findings in [6].

As can be expected, LR performs better when it has a higher input dimensionality, as additional, less useful signals do not interfere with useful signals. On the other hand, RC combines its input signals and needs a good balance between the input dimensionality and the actual useful information these signals contain.

If we take a look at `imu` and `joints` in Table 5.3, we notice a poor performance compared to the other results. These are the only signal

Tab. 5.4: Train and test results for comparison of RC and ELM with varying number of nodes in the hidden layer (in percentage, more is better). Signal selections touch and minimal are tested, using the entire dataset.

		TOUCH			MINIMAL		
		50N	100N	200N	50N	100N	200N
TRAIN	LR	64.45			82.28		
	ELM	83.04	84.19	85.75	87.32	91.84	94.38
	RC	88.17	90.40	92.99	88.27	91.73	95.01
TEST	LR	57.89			67.36		
	ELM	72.83	72.47	75.52	74.58	79.82	80.80
	RC	80.30	83.04	85.19	74.09	77.28	81.43

selections without the tactile sensors. `leg` and `agt` are exactly the same as `joints` and `imu` with the exclusion of the tactile sensors (and the inclusion of the magnetometer in `imu`). This clearly indicates the importance of the tactile sensors for supervised terrain classification. RC even achieves the best result solely using the tactile sensors.

5.4.1.2 Selection of the Processing System

In order to compare the different supervised systems discussed before (LR, ELM and RC), we compare their performance for two sensor combinations: `touch` and `minimal`. We picked `minimal` because it seemed to strike a good balance between dimensionality and performance across both techniques. Since RC achieved its best score on the sensor combination `touch`, it was added as well. From the results in Table 5.4 it is clear that the non-linearities introduced by expanding the input sensors are necessary for a good result, as linear regression has over 10% more misclassifications then any other technique with 100 hidden nodes. The results also show that given enough input signals (`minimal`: 18 dimensions), RC and ELM achieve virtually the same performance. Using only four tactile sensors on the other hand, requires a system with recurrent nodes, as RC outperforms ELM

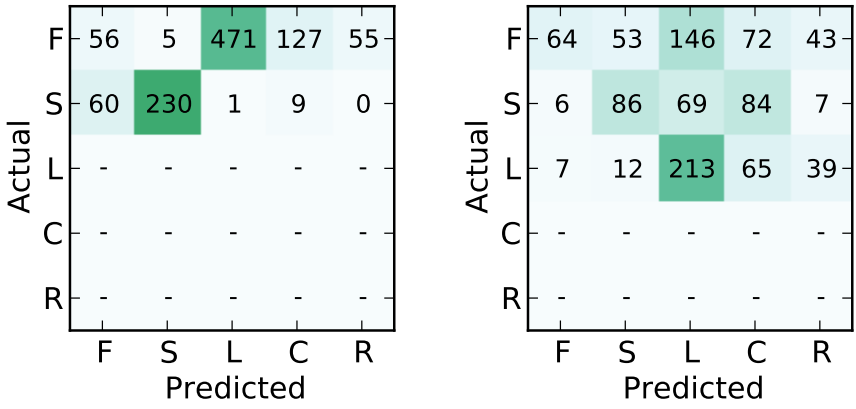


Fig. 5.3: Reservoir computing confusion matrix for testing gait frequency (left) and gait (right) generalization.

roughly 10%, and at the same time achieves the best result overall, even while having less input information.

5.4.1.3 Generalizability

Lastly we investigate the generalizability of the methods to other gaits or gait frequencies. We train the system on the single gait 'bound right' at the single frequency 1Hz. Subsequently we test it on other frequencies of the same gait on the one hand and other gaits at the same frequency on the other hand. Figure 5.3 depicts both these approaches for RC. The extent of this experiment is limited by the available trials in the dataset, as can be seen in Table 5.1. The dataset contains only three terrains for the gait generalization and two terrains for the gait frequency.

Looking at the gait frequency we notice a high performance for styrofoam (S) but a complete misclassification for foil (F), with a similar result for LR. Looking back at table 5.1 we notice that only 2 trials on foil are available for training, while there are 13 trials on styrofoam (only 5 used during training). This explains the poor performance for foil, meaning the performance for styrofoam might indicate a possible generalization to different frequencies.

Gait generalization on the other hand seems less feasible. The result

for foil can again be attributed to its limited train set, but styrofoam has an equally poor performance. Only linoleum (L) achieves a fairly decent classification. If we would have a larger dataset and use multiple gaits during training, gait generalization might be possible. Only a single gait and gait frequency on the other hand does not carry enough information to correctly classify new gaits.

5.4.2 Unsupervised

5.4.2.1 Sensor selection

Table 5.5 depicts the results for the experiment with the unsupervised techniques, similarly as the experiment with supervised techniques. The entire dataset is used and PCA is applied for compacting the input signals.

Even though SFA2 and RC-SFA are mostly the same system, the different expansion of the input signals clearly has a large impact on the performance. This indicates that the way reservoirs expand the dynamics of the system is beneficial for classifying terrains. SFA2 performs relatively close to a random terrain sensor (25%), while RC-SFA performs about 10 to 20% better. The signal selections `imu` and `joints` seem to perform rather well, unlike with the supervised techniques. Solely using the tactile sensors still achieves a decent performance, but SFA seems to benefit from more input sensors.

5.4.2.2 Selection of the Processing System

Similarly as in section 5.4.1.2, we investigate the importance of the recurrent connections in the reservoir. To do this, we compare the results of reservoir computing with leaky ELM's. In Table 5.6, RC-SFA is compared with ELM-SFA using the sensor group `minimal` and the bound right gait at all frequencies.

So with unsupervised learning techniques, the same conclusion of the supervised techniques holds as well. Here, the recurrent connections do not seem to make a significant difference either on the `minimal` sensor group. Note on the other hand the decent results for unsuper-

Tab. 5.5: Train and test results for eight signal selections (in percentage, more is better). The entire table has a single color gradient: maximal value is green, minimal is red, average of the two is yellow.

	TRAIN		TEST	
	SFA2	RC-SFA	SFA2	RC-SFA
useful	58.11	79.75	38.94	51.69
reduced	60.98	81.03	36.27	58.28
minimal	61.72	79.26	37.90	57.29
leg	53.71	72.25	33.82	58.33
agt	51.49	75.21	26.99	52.75
imu	44.45	70.83	29.16	55.29
joints	38.22	71.15	27.50	54.58
touch	48.09	75.53	35.97	50.29

Tab. 5.6: Train and test results for comparison of RC-SFA and ELM-SFA with varying number of nodes in the hidden layer (in percentage, more is better). The bound right gait at all frequencies were used.

		MINIMAL		
		50N	100N	200N
TRAIN	SFA2	61.72		
	ELM-SFA	81.92	82.93	86.24
	RC-SFA	83.97	82.71	86.60
TEST	SFA2	37.90		
	ELM-SFA	63.16	70.31	72.64
	RC-SFA	65.64	68.62	71.92

vised classification when only using a single gait, as opposed to the results of the complete dataset from Table 5.5.

5.5 Conclusion

In this paper, we showed that a limited but appropriate selection of input sensors is sufficient to perform terrain classification on our legged robot. We have demonstrated that good results are achievable with nearly all methods tested, using the combination of a 6 DoF IMU, joint angle sensors and tactile sensors on the feet. The fact that linear regression performed badly on the data, indicates the importance of adding non-linearities. For these non-linearities, we found that reservoirs perform better than quadratically expanding, as is demonstrated by the unsupervised classification method. For supervised learning, reservoir computing led to drastic better performance when using few sensors, which indicates that the richness of the non-linear temporal expansion is beneficial for classification with less information. The memory capacity seems to be an important element, as the filtering used in the ELMs proved to be insufficient.

How generally applicable this conclusion is, cannot be reliably determined from this study, since only one robot was used to obtain the data. We note that this observation was reached on all tested terrains and gaits. Given sufficient data in the train set, it was possible to classify terrains even at unseen frequencies. However, when trained on a single gait, the methods studied here were not very effective at generalizing to unseen gaits.

We want to conclude that the challenge is to find an appropriate set of input sensors to classify terrains in quadruped robots. On the Puppy II robot, we found that a limited set of sensor inputs were enough to perform good quality terrain classification, when using methods which take into account that there are underlying dynamics and non-linearities in the system. We found that reservoir computing is a good way to take these dynamics and non-linearities into account, since the fading memory introduced by the recurrent connections between the nodes improved performance when few sensors are available, compared to other similar techniques.

Acknowledgments

The authors would like to thank Matej Hoffmann of the University of Zurich again for lending us the dataset obtained from the Puppy II robot. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 – Challenge 2 – Cognitive Systems, Interaction, Robotics – under grant agreement No 248311 - AMARSi.

Bibliography

[1] J. Degrave, R. Van Cauwenbergh, F. wyffels, T. Waegeman, and B. Schrauwen, "Terrain classification for a quadruped robot," in *International conference on machine learning and applications (ICMLA)*. IEEE, 2013, pp. 185–190.

[2] J. Buchli, M. Kalakrishnan, M. Mistry, P. Pastor, and S. Schaal, "Compliant quadruped locomotion over rough terrain," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, oct. 2009, pp. 814 –820.

[3] C. Plagemann, S. Mischke, S. Prentice, K. Kersting, N. Roy, and W. Burgard, "Learning predictive terrain models for legged robot locomotion," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, sept. 2008, pp. 3545 – 3552.

[4] P. Giguere, G. Dudek, C. Prahacs, and S. Saunderson, "Environment identification for a running robot using inertial and actuator cues," *Proceedings of Robotics Science and System*, August 2006.

[5] M. Hoepflinger, C. Remy, M. Hutter, L. Spinello, and R. Siegwart, "Haptic terrain classification for legged robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, may 2010, pp. 2828 –2833.

[6] M. Hoffmann, N. M. Schmidt, R. Pfeifer, A. K. Engel, and A. Maye, "Using sensorimotor contingencies for terrain discrimination and adaptive walking behavior in the quadruped robot

puppy," in *From Animals to Animats 12*. Springer, 2012, pp. 54–64.

[7] P. Giguere and G. Dudek, "Clustering sensor data for autonomous terrain identification using time-dependency," *Autonomous Robots*, vol. 26, no. 2-3, pp. 171–186, Apr. 2009.

[8] P. Giguere, "Unsupervised learning for mobile robot terrain classification," 2009.

[9] H. Guang-Bin, Z. Qin-Yu, and S. Chee-Kheong, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489 – 501, 2006.

[10] D. Verstraeten, B. Schrauwen, M. d'Haene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Networks*, vol. 20, no. 3, pp. 391–403, 2007.

[11] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[12] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[13] J. J. Steil, "Backpropagation-decorrelation: Online recurrent learning with O(N) complexity," in *IEEE International Joint Conference on Neural Networks*, vol. 2. IEEE, 2004, pp. 843–848.

[14] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, "Optimization and applications of echo state networks with leaky-integrator neurons," *Neural Networks*, vol. 20, no. 3, pp. 335–352, 2007.

[15] K. Caluwaerts, F. wyffels, S. Dieleman, and B. Schrauwen, "The spectral radius remains a valid indicator of the echo state property for large reservoirs," in *International Joint Conference on Neural Networks (IJCNN)*, 2013.

[16] M. Lukoševičius and H. Jaeger, "Survey: Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[17] M. Hermans and B. Schrauwen, "Memory in linear recurrent neural networks in continuous time," *Neural Networks*, vol. 23, no. 3, pp. 341–355, 2010.

[18] E. Antonelo and B. Schrauwen, "Learning slow features with reservoir computing for biologically-inspired robot localization," *Neural Networks*, vol. 25, pp. 178–190, 2012.

[19] E. I. Moser, E. Kropff, and M.-B. Moser, "Place cells, grid cells, and the brain's spatial representation system," *Annual Reviews Neuroscience*, vol. 31, pp. 69–89, 2008.

[20] L. Wiskott and T. J. Sejnowski, "Slow feature analysis: Unsupervised learning of invariances," *Neural computation*, vol. 14, no. 4, pp. 715–770, 2002.

[21] P. Comon, "Independent component analysis, a new concept?" *Signal Processing*, vol. 36, no. 3, pp. 287 – 314, 1994.

[22] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural Networks*, vol. 13, no. 4-5, pp. 411 – 430, 2000.

[23] N. Hansen, "The CMA evolution strategy: a comparing review," in *Towards a new evolutionary computation. Advances on estimation of distribution algorithms.* Springer, 2006, pp. 75–102.

[24] N. M. Schmidt, M. Hoffmann, K. Nakajima, and R. Pfeifer, "Bootstrapping perception using information theory: Case studies in a quadruped robot running on different grounds," *Advances in Complex Systems*, pp. 125 – 138, 2013.

6

Developing an Embodied Gait on a Compliant Quadrupedal Robot

J. Degrave, K. Caluwaerts, J. Dambre, and F. wyffels, "Developing an embodied gait on a compliant quadrupedal robot," in *IEEE International Conference on Intelligent Robots and Systems*. IEEE, 2015, pp. 4486–4491

⋆⋆⋆

Incorporating the body dynamics of compliant robots into their controller architectures can drastically reduce the complexity of locomotion control by exploiting the morphological computation going on in the body of the robot. An extreme version of this embodied control principle was demonstrated in highly compliant tensegrity robots, for which stable gait generation was achieved by using only optimized linear feedback from the robot's sensors to its actuators. The morphology of quadrupedal robots has previously been used in chapter 5 for sensing and for control of a compliant spine, but not for gait generation. In this chapter, we apply embodied control to the compliant, quadrupedal Oncilla robot. As initial experiments indicated that mere linear feedback does not suffice, we explore the minimal requirements for robust gait generation in terms of memory and nonlinear complexity. Our results show that a memory-less feedback controller can generate a stable trot by learning the desired nonlinear relation between the input and the output signals. We believe this method can provide a robust tool for transferring knowledge from open loop to closed loop control on compliant robots.

6.1 Introduction

Compliant robots have steadily been gaining interest due to their increased ability to interact with the environment and unexpected disturbances. One way to implement compliance is by controlling impedance and joint torque [2], often referred to as active compliance. Successful implementations of this approach can be found in the well-known Big-Dog quadrupedal robot [3] or the SARCOS humanoid [4].However, robots with active compliance rely on inticrate software solutions and often complex sensors to make stiff actuation modules compliant. To reduce the complexity of the sensing and control modules, and motivated by energy efficiency and a safer robot-human-world interaction, recent trends in robotics tend to use compliant actuation modules, which can have either fixed or regulated compliance [5, 6]. The evolution towards compliance also extends to other parts of the robot as flexible materials are being used for structural parts of the robot. Examples of such robots are the quadrupedal robots Oncilla [7] and StarlETH [8], the i-HY hand, which consists of flexible fingers that can manipulate a wide variety of objects [9], and the tensegrity robot ReCTeR [10].

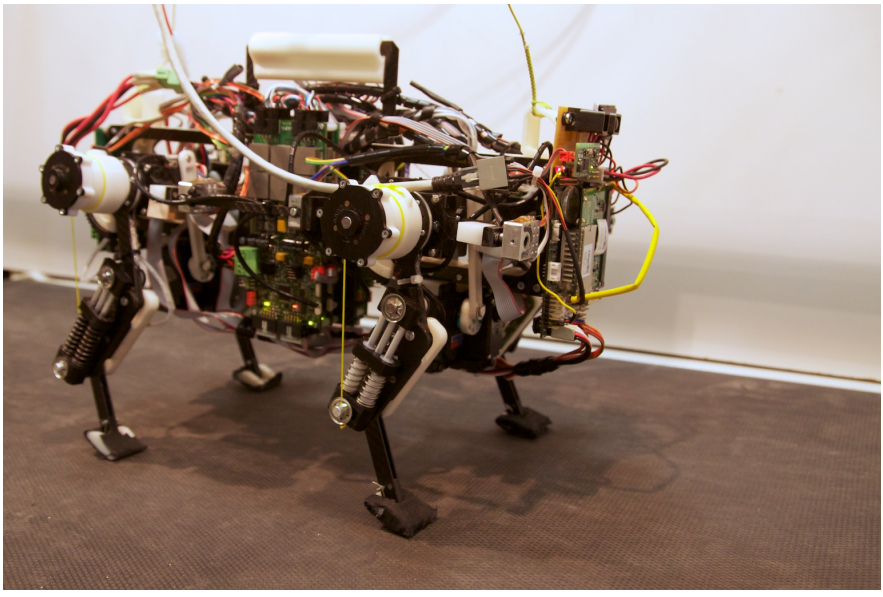


Fig. 6.1: The Oncilla robot on the treadmill.

Unfortunately, compliant robots are harder to control due to the increased non-linear behaviour of the elastic elements. For this reason, the focus on the control algorithms has been shifted towards the morphology. The idea is that control tasks, such as locomotion control, can be partially outsourced to the compliant body elements and their interaction with the environment. This concept is known as morphological computation [11]. Recently, it was shown that certain compliant structures such as spring-mass networks have universal computing power [12]. This is highly related to the field called *physical reservoir computing*, in which the principles of reservoir computing [13, 14, 15] are applied to physical systems. While physical implementations exist ranging from a water bucket [16] to integrated photonics devices [17], robotic implementations are rare. Nevertheless, recent work [10, 18] illustrated that locomotion control can be outsourced to the body of a tensegrity robot: a structure composed of compression elements held together by a compliant tensile network. Stable gait generation was achieved by using only optimized linear feedback from the robot's stretch sensors to its actuator control signals.

In this paper, we test the same principle on a much less compliant robot, the quadrupedal robot Oncilla [7], shown in Fig. 6.1. In earlier research, the morphology of quadruped robots has been used for sensing [19] and for control of a compliant spine [20], but never for gait generation. We not only show that robust locomotion control by a simple mapping from the rotary encoders in the motors is possible, but we also investigate under which conditions this can be achieved.

When physical reservoir computing is applied to robotics, the robot body is a highly specific dynamical system, to which, in general, existing proofs of computational universality do not apply. This implies that a mismatch can exist between the actually observed robot states and the dynamical transformations that are required for the task. It has been shown that, when keeping a fixed number of observed states (i.e., sensor readouts), there is a trade-of between memory and non-linear computing power [21]. In this paper, we propose the introduction of an additional transformation between the physical body and the linear combination layer. We investigate the requirements to such a transformation by tuning two dimensions of its complexity: memory and nonlinearity. By doing so, we can investigate which dynamics are desired in order to outsource locomotion control for the

Oncilla quadrupedal robot platform.

The remainder of this paper is structured as follows. We first describe our robot and describe the control architecture we have used. We subsequently describe the experimental setup and present and discuss the results obtained with our approach on the Oncilla robot. We end our paper by presenting our conclusions.

6.2 The Oncilla Robot

The quadrupedal Oncilla robot is the compliant platform used in this work, see Fig. 6.1. Each of the robot's legs has a three-segmented pantographic system to achieve similar dynamical to those of felines. The robot has actuated hip and shoulder joints that can perform both an abduction and a flexion motion. The knee joint is actuated with a short thread so it can only perform actuated flexion, while an opposite pushing spring does the extension. For a complete overview of this robot platform, see [7].

The robot has 12 actuated degrees of freedom and a variety of sensors. In this paper, we only use the 8 rotary encoders on the hip and knee joints of the robot. These sensors are chosen as they are found in nearly all quadrupedal robots, which will allow for a broader application of this approach. The time between consecutive updates of the sensor readings and motor actuations is on average $\Delta t \approx 8.2\,\text{ms}$. However, this period can vary as much as 15 % depending on the computational complexity of the controller. The robot can operate fully autonomously, but for the sake of this paper, we power the robot with a power cable and process the signals on a remote computer.

In order to run experiments without being constrained by the space available in our lab, we put our robot on a treadmill. The robot is equipped with a distance sensor on its head, such that the treadmill can adjust speed to keep the robot in the middle of the treadmill, as shown in Fig. 6.2. An assistant sits next to the setup to intervene when the robot would put its own safety in jeopardy.

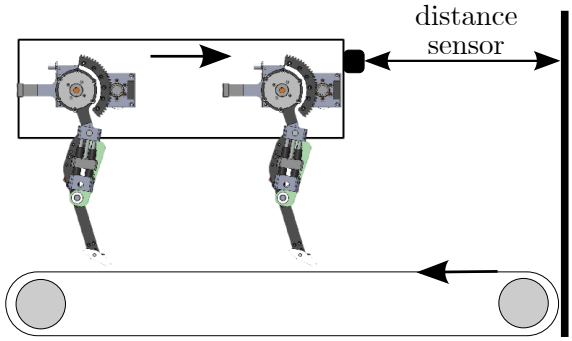


Fig. 6.2: The Oncilla robot on the treadmill. The distance sensor measures the distance from the robot to the front of the treadmill. The treadmill then adjusts speed to keep the robot in the center of the treadmill. This setup allows the robot to run for minutes at end while not having to deviate from a straight line.

6.3 Controller Architecture

6.3.1 Embodied Computation

In earlier work [10, 18], it was shown that for a tensegrity robot, a linear transformation from the sensor signals to the motor signals was enough to generate stable locomotion. The idea behind this is that the body of the robot itself has computing power, and that this power is being harvested by using it as a reservoir. This *embodiment* of computation allows the robot to generate stable locomotion without the requirement to explicitly use the state of its compliant elements in a digital control algorithm.

On our robot however, we found that we could indeed generate a gait this way, but that it was not stable and did not always return to its limit cycle. In other words, the robot's internal dynamics in response to the environment do not exactly match the required dynamics for stable gait generation. However, the fact that they suffice to generate a close but unstable approximation indicates that the mismatch is not very large.

Therefore, we propose to digitally add additional transformations to

the sensor signals. In this way, the computations are still partially embodied in the morphology of the robot. In this setting, we want to quantify the minimal complexity of these transformations as expressed by their memory and nonlinear complexity. The next sections describe its parts in more detail.

6.3.2 Linear Transformation

The aim of the linear transformation is to find the $M \times N$ transformation matrix $\boldsymbol{W}$ that optimally maps the $N \times 1$ vector of the N normalized input signals $\boldsymbol{x}$ to the $M \times 1$ vector of the M output signal $\hat{\boldsymbol{y}}$:

$$\hat{\boldsymbol{y}} = \boldsymbol{W} \cdot \boldsymbol{x}.$$

Optimality is defined as the minimisation of the mean squared error (*MSE*) between the output signals $\hat{\boldsymbol{y}}$ and the target output signals $\boldsymbol{y}$. This can be achieved by using linear regression:

$$\boldsymbol{W} = \boldsymbol{y} \cdot \boldsymbol{x}^{+}$$

To achieve the right bias, we add a constant signal to the inputs $\boldsymbol{x}$.

This approach is limited to one-shot learning. In order to continue optimizing this relation while running, we will use the recursive least squares (RLS) algorithm [22], an online method for linear regression. In what follows, we introduce the vector $\boldsymbol{x_{rls}}$ to indicate the inputs for the RLS-algorithm, because in the remainder of this work these will be transformed versions of the sensor outputs $\boldsymbol{x}$. In RLS, the weight matrix $\boldsymbol{W_{rls}}$ is updated at each time step according to the following equations:

$$\boldsymbol{L_{rls}}(t) = \frac{\boldsymbol{P_{rls}}(t) \cdot \boldsymbol{x_{rls}}(t)}{1 + \boldsymbol{x}_{\boldsymbol{rls}}^{\mathrm{T}}(t) \cdot \boldsymbol{P_{rls}}(t) \cdot \boldsymbol{x_{rls}}(t)}$$

$$\boldsymbol{P_{rls}}(t + \Delta T) = \boldsymbol{P_{rls}}(t) - \frac{\boldsymbol{P_{rls}}(t) \cdot \boldsymbol{x_{rls}}(t) \cdot \boldsymbol{x}_{\boldsymbol{rls}}^{\mathrm{T}}(t) \cdot \boldsymbol{P_{rls}}(t)}{1 + \boldsymbol{x}_{\boldsymbol{rls}}^{\mathrm{T}}(t) \cdot \boldsymbol{P_{rls}}(t) \cdot \boldsymbol{x_{rls}}(t)}$$

$$\boldsymbol{e_{rls}}(t) = \boldsymbol{y}(t) - \boldsymbol{W_{rls}}(t - \Delta t) \cdot \boldsymbol{x_{rls}}(t)$$

$$\boldsymbol{W_{rls}}(t) = \boldsymbol{W_{rls}}(t - \Delta t) + \boldsymbol{e_{rls}}(t) \cdot \boldsymbol{L}_{\boldsymbol{rls}}^{\mathrm{T}}(t).$$

Here, Δt is the controller time step. $\boldsymbol{P_{rls}}$ is the $N \times N$ precision matrix, which is initialized with the identity matrix, such that the noise on

the input signals is initially assumed uncorrelated. $\boldsymbol{e_{rls}}$ represents the $M \times 1$ a priori error vector. $\boldsymbol{W_{rls}}$ is the matrix that represents the linear transformation, which is initialized with zeros.

6.3.3 Adding Non-Linear Dynamics

Between the robot body and the linear transformation, we now add an additional transformation layer in order to increase the richness of signals received by the linear transformation. In order to be able to separately explore the need for memory and nonlinearity, we introduce two separate modules: a nonlinear layer and a memory buffer.

The nonlinear transformations are generated by introducing a hidden layer of H nonlinear neurons, each of which receives a random mixture of the sensor signals (again augmented with a constant bias signal):

$$\boldsymbol{x_{nl}}(t) = \tanh(\boldsymbol{W}_{hidden} \cdot \boldsymbol{x}(t) + \boldsymbol{w}_{bias}).$$

This technique is known as Extreme Learning Machines (ELM) [23]. In our paper, we initialize all elements in the matrix $\boldsymbol{W}_{hidden}$ and the vector $\boldsymbol{w}_{bias}$ by sampling them from the standard normal distribution. These elements are not optimized.

The memory buffer of length B is added to each nonlinearly transformed signal, such that the RLS algorithm obtains direct access to the signals from previous time steps in $\boldsymbol{x_{rls}}(t)$. $\boldsymbol{x_{rls}}$ thus contains all signal values from a small time window in the past. This allows us to explore a further richness of the dynamics which were added by the morphology of the robot:

$$\boldsymbol{x_{rls}}(t) = \begin{Bmatrix} \boldsymbol{x_{nl}}(t) \\ \boldsymbol{x_{nl}}(t - \Delta t) \\ \boldsymbol{x_{nl}}(t - 2\Delta t) \\ \vdots \\ \boldsymbol{x_{nl}}(t - B\Delta t) \end{Bmatrix}.$$

The resulting detailed controller architecture is schematically represented in Figure 6.3.

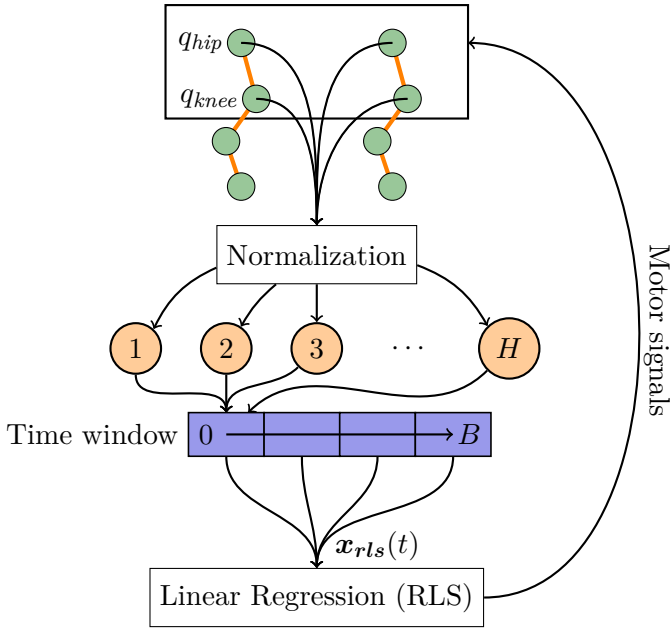


Fig. 6.3: A schematic of the control system. The signals measured in the motor encoders are first normalized, are then send through a layer of hidden, untrained neurons, the outputs of these neurons are buffered and a linear transformation is performed on these buffered signals to generate the motor signals. It is only this linear transformation which is learned through a linear regression method (RLS).

6.4 Experimental Setup

We want our robot to realise a stable gait based on the feedback from the 8 rotary encoders. Using these sensors, the proposed controller must be trained to derive a motor command which is sent to the end-effectors. In sequential operation, this should result in the robot moving with a stable gait. The training procedure outlined in this section enables the controller to discover the relation between the received sensor signals and the output it needs to generate at that moment.

As target signals, we use motor signals for stable gaits resulting from previous work [24, 25]. We have the robot trot at a frequency of 1.7 Hz, corresponding to a speed of about 0.76 m/s. Before each experiment, the robot runs for 5 seconds with this gait, using the desired signal as input, in order to reach steady state. During these 5 seconds, we measure the mean and the standard deviation of the sensor signals. These are used to normalize the input signals such that they have a mean of 0 and a standard deviation of 1. After normalization, we add Gaussian noise with an amplitude of 0.01 to each input signal for regularization during training.

In the first training phase, the linear combination is optimized using RLS. As a result of this training, the control system finds a relation between the input and the output signals, but it fails to find a stable attractor. Every time the robot has a small error in the output signal, this error is reflected in the input signals of the next time step. Since the controller has never learned to handle those errors, they accumulate and destabilize the attractor.

For the controller to learn to deal with its own errors, we add a second training phase, with RLS still active. In this stage, the output signals sent to the motors are mixtures of the target signals and the signals generated by the linear transformation [18]. The fraction of the target signals is reduced over time until it becomes zero. After this phase, the RLS learning is switched off and the resulting gait is evaluated.

We thus split up the learning process into multiple phases:

1. **The normalization phase**: We wait for transient effects

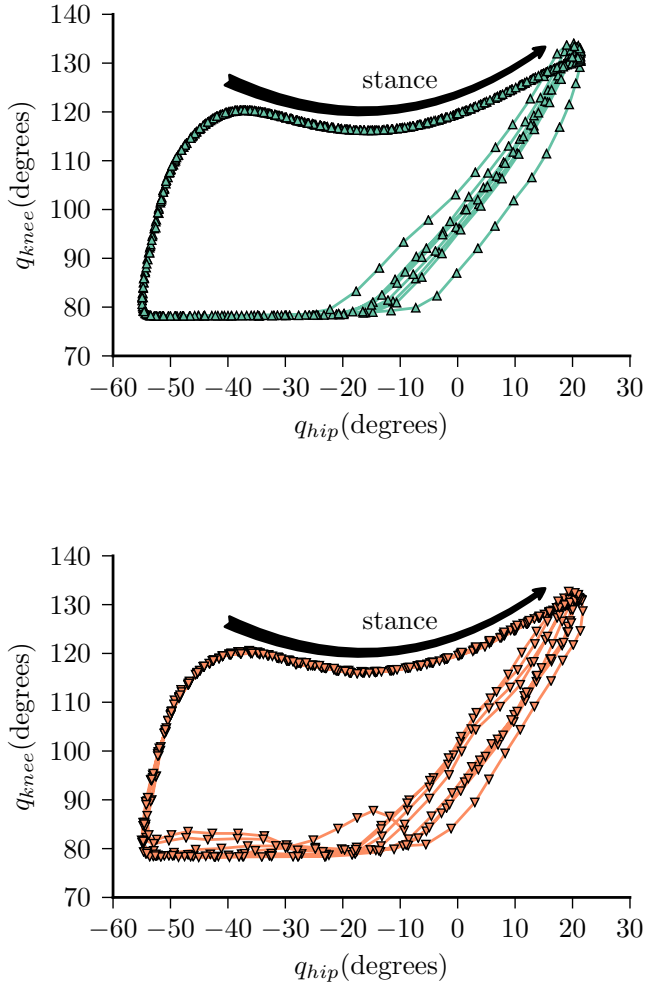
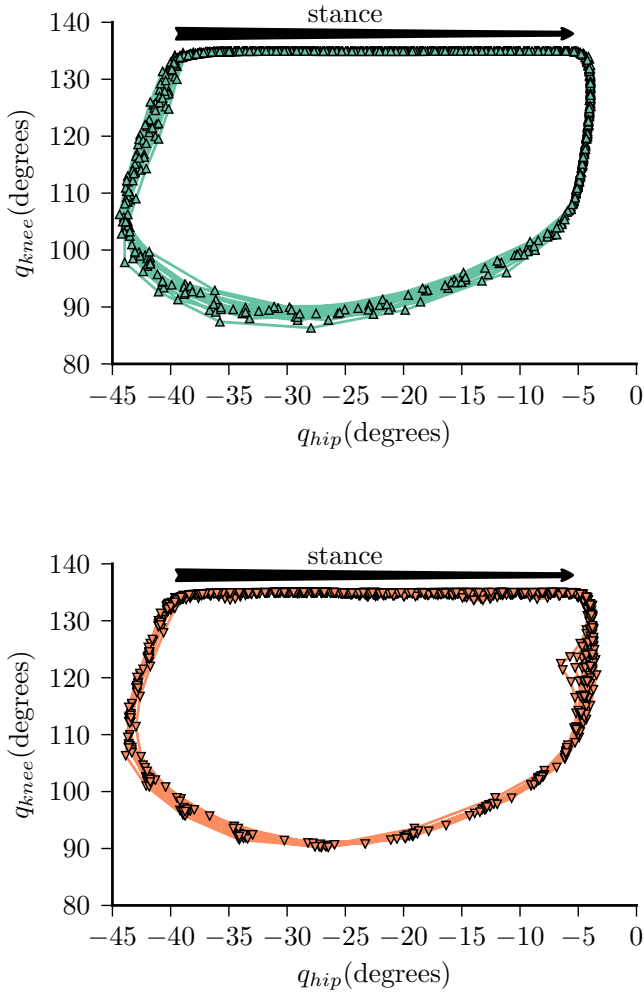


Fig. 6.4: On the top (▵), the trajectory of the robot's right hind leg in the joint space is shown while the robot is performing a trotting gait, with q_{hip} and q_{knee} being the angle measured over time by the hip and knee encoder respectively. These were recorded during the first 5 seconds of the first training stage. A higher angle means the leg is moved to the front or the knee is extended. On the bottom (▿), the same trajectory is shown, but with the learned controller, with $B = 5$ and $H = 50$ on top and $B = 12$ and $H = 50$ on the bottom. These were recorded between 25 and 30 seconds into the running stage.



On the top (▲), the trajectory of the robot's right hind leg in the joint space is shown while the robot is performing a walking gait, with q_{hip} and q_{knee} being the angle measured over time by the hip and knee encoder respectively. These were recorded during the first 5 seconds of the first training stage. A higher angle means the leg is moved to the front or the knee is extended. On the bottom (▼), the same trajectory is shown, but with the learned controller, with $B = 5$ and $H = 50$ on top and $B = 12$ and $H = 50$ on the bottom. These were recorded between 25 and 30 seconds into the running stage.

caused by starting from standstill to fade out, and when we record the average and variance of each sensor to normalize them. This stage takes 5 s.

2. **The first training phase**: We send the teacher signal to the motors, and use RLS to learn the relation between these outputs and the normalized inputs from the sensors. This stage takes 10 s, unless noted differently.

3. **The second training phase**: The motor signals are mixed between the teacher signal and the signals generated by the linear transformation. The RLS-algorithm still updates $\boldsymbol{W_{rls}}$. This stage takes 10 s, unless noted differently.

4. **The running phase**: The robot stops optimizing the linear transformation, but continues to run and where we test the stability of the attractor.

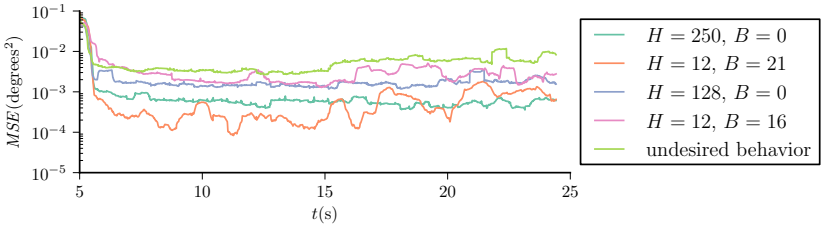


Fig. 6.6: Evolution of the MSE during training, in which the noise has been filtered by a moving average filter with the length of one gait period: without time window (turkoise and blue), with reduced number of hidden neurons (orange and pink) and a result in which the controller failed to find a stable attractor because it did not have enough signals (light green).

In the controller described in the previous section, there are three parameters: the number of hidden neurons H, the number of time steps in the time window B and the training time t_{train}. As the hidden neuron layer is the only nonlinear part of the controller, the number of hidden neurons H gives an indication of the amount of nonlinearity that is needed in the system. The number of time steps in the time window B is a measure for how much the system depends on time information. Finally, the training time t_{train} is a measure for the complexity of the learning task. When a stable attractor is

found quickly, this suggests that we may be able reduce the number of input signals for linear regression, at the cost of a longer training time. Vice versa, if it takes long to find a stable attractor, the training time could probably be reduced by adding more input signals to the linear regression.

In order to evaluate the importance of each of these three parameters, we conducted three experiments. These were aimed at determining the interaction between these parameters and at finding their minimal values before the attractors become unstable. We are however limited by computation speed. The RLS-algorithm scales with $O(N^2)$, with N being the number of signals it receives. In our case, this is equal to $(B+1)H+1$. When this number becomes too large, Δt needs to increase to allow enough time for computations, which is bad for the accuracy of the movements of the robot.

We identify three types of undesired behavior the trained controller can display:

- the output signals die out and the robot motion freezes, because the attractor has a stable, fixed point equilibrium;
- the robot moves erratically because the attractor has the wrong limit cycle or no limit cycle;
- the robot is stable while in gait, but cannot return to the limit cycle once it has left it, because the attractor has a limit cycle for which the basin of attraction is too small.

In this paper, we therefore look for the minimal values of the parameters in our control system that allow our robot to run stably for 30 seconds, and to return to its gait after the motors have been powered off and are powered back on. This way we test the controller's robustness to these three problems.

6.5 Results

To evaluate our approach, we optimized a controller with $H = 50$ hidden nodes and a time window B of 5 time steps (44 ms or 7.5 %

of the gait period). For these settings, an attractor was found that generated a stable gait and was able to return to its limit cycle after stopping. The resulting attractor is depicted in Fig. 6.4.

In order to prove that this result is reproducible on different setups, we trained the Oncilla robot to perform a walking gait. For this situation, we again used $H = 50$ hidden nodes, but we had to increase the time window B to 12 time steps (103 ms or 17.5 % of the gait period). The resulting attractor is shown in Fig. 6.4. We also needed to increase the length of the first and second training stage to 30 s.

The fact that we needed to increase both B and t_{train} is explained by the increased complexity of a walking gait. In this gait, each leg has a different phase which means that there is less dependence between the motor signals that need to be generated. Therefore, we had to increase the number of inputs to the RLS layer to the maximum we could compute in real time. Additionally, we needed to increase the training time to find the proper relation between the inputs and the outputs.

Since our approach was reproducible, we consequently tried to reduce the parameters for the easier trot gait to identify the point at which the controller fails to find a stable attractor. We first reduced the number of hidden neurons H. It makes little sense to have $H < 12$, since we have 12 independent outputs to generate. We found that with 12 hidden neurons, we needed B to be at least 16 time steps (115 ms or 20 % of the gait period) for a stable gait.

Secondly, we removed the buffer ($B = 0$), and searched for the minimal number of hidden neurons required for finding a stable attractor. We found that without a time window, we need at least $H = 128 \pm 32$ hidden neurons.

Thirdly, we tried to reduce the training time. For this, we used a controller without a buffer and with $H = 250$ hidden neurons. We found that 1.18 s, or two gait periods are enough for both the first and second training stage, or 2.36 s in total.

6.6 Discussion

We can explain the observations in the previous section by looking at the *MSE* during the optimization process. At each time step, we can evaluate the difference between the output generated by the linear transformation and the output of the teacher signal. In Fig. 6.6 we plot the evolution of the *MSE* over time for different optimizations, filtered by a moving average with a time window of one gait period.

The figure shows that a lower number of signals $N = (B + 1)H + 1$ received by the RLS algorithm results in a higher value of the *MSE* during training. Moreover, when the *MSE* is too high during training, the system fails to find a stable attractor. This implies that we can use the *MSE* as a dynamic stopping criterion for training. As long as the it is too high, the controller will not be able to compensate its own errors which accumulate over time. The fact that we were able to considerably reduce the training time also supports this. The figure also shows that the *MSE* drops rapidly during the first seconds.

When we compare the relative importance of memory and nonlinearity, we find that both contribute to the performance of the controller, but that they are more or less interchangeable. This can possibly be explained by the nonlinearity of the body dynamics, which results in sensor signals being to some extent correlated to nonlinearly transformed versions of their own history. The setup of our research did not allow to find a significant difference in relative importance between their contribution. Since Δt can vary as much as 15 % and that $\boldsymbol{W}_{hidden}$ and $\boldsymbol{w}_{bias}$ were arbitrarily sampled from a standard normal distribution, both contributions could be further optimized. However, this would unnecessarily complicate the setup. Moreover, it is an indication that this approach is robust and we believe that it could be more broadly applicable.

We want to stress that allthough the controller might seem complex, it does not eliminate the existence of morphological computation in our setup. In the rudementary case where $H = 128, B = 0$, the linear regression might receive 129 linearly independent signals. These signals are however non-linearly dependent, and have at most 8 statistically independent components, namely the 8 sensor signals. We have undergoing research to show more clearly that morphological computing

plays an essential role in this approach.

6.7 Conclusion

In this paper, we demonstrated how an embodied control system with memory-less nonlinear feedback can generate a dynamically balanced trot on a compliant quadrupedal robot. Our feedback controller, based on extreme learning machines, learns the desired relation between the input and the output signals in the time span of only a couple of strides.

We have shown that this method can be extended to other gaits, such as a walk, when increasing the training time and the model complexity. The incorporation of either additional memory or additional non-linearities contribute approximately equally to the controller performance. The parameter that mainly determines the performance is the number of signals that is fed into the the linear transformation.

As our controller was trained directly on the actual robot, we did not have to rely on a simulation model, which is often unreliable on a compliant robot. In addition, the controller optimisation was fast, happened entirely online and automatically. We believe that the proposed method can provide a useful tool for transferring knowledge from open loop to closed loop control.

Bibliography

[1] J. Degrave, K. Caluwaerts, J. Dambre, and F. wyffels, "Developing an embodied gait on a compliant quadrupedal robot," in *IEEE International Conference on Intelligent Robots and Systems*. IEEE, 2015, pp. 4486–4491.

[2] A. Albu-Schäffer, C. Ott, and G. Hirzinger, "A unified passivity-based control framework for position, torque and impedance control of flexible joint robots," *The International Journal of Robotics Research*, vol. 26, no. 1, pp. 23–39, 2007.

[3] R. Playter, M. Buehler, and M. Raibert, "Bigdog," in *Defense*

and Security Symposium. International Society for Optics and Photonics, 2006, pp. 62 302O–62 302O.

[4] S. Hyon, J. G. Hale, and G. Cheng, "Full-body compliant human–humanoid interaction: balancing in the presence of unknown external forces," *IEEE Transactions on Robotics*, vol. 23, no. 5, pp. 884–898, 2007.

[5] R. v. Ham, T. G. Sugar, B. Vanderborght, K. W. Hollander, and D. Lefeber, "Compliant actuator designs," *IEEE Robotics & Automation Magazine*, vol. 16, no. 3, pp. 81–94, 2009.

[6] B. Vanderborght, A. Albu-Schäffer, A. Bicchi, E. Burdet, D. G. Caldwell, R. Carloni, M. Catalano, O. Eiberger, W. Friedl, G. Ganesh *et al.*, "Variable impedance actuators: A review," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1601–1614, 2013.

[7] A. Sproewitz, L. Kuechler, A. Tuleu, M. Ajallooeian, M. D'Haene, R. Moeckel, and A. J. Ijspeert, "Oncilla robot: a light-weight bioinspired quadruped robot for fast locomotion in rough terrain," in *Proceedings of the Fifth International Symposium on Adaptive Motion on Animals and Machines*, 2011.

[8] M. Hutter, C. Gehring, M. Bloesch, M. Hoepflinger, C. D. Remy, and R. Siegwart, "Starleth: A compliant quadrupedal robot for fast, efficient, and versatile locomotion," in *International Conference on Climbing and Walking Robot (CLAWAR)*, no. 15, 2012.

[9] L. U. Odhner, L. P. Jentoft, M. R. Claffee, N. Corson, Y. Tenzer, R. R. Ma, M. Buehler, R. Kohout, R. D. Howe, and A. M. Dollar, "A compliant, underactuated hand for robust manipulation," *The International Journal of Robotics Research*, vol. 33, no. 5, pp. 736–752, 2014.

[10] K. Caluwaerts, J. Despraz, A. Işçen, A. P. Sabelhaus, J. Bruce, B. Schrauwen, and V. SunSpiral, "Design and control of compliant tensegrity robots through simulation and hardware validation," *Journal of The Royal Society Interface*, vol. 11, no. 98, 2014.

[11] R. Pfeifer and F. Iida, "Morphological computation: Connecting body, brain and environment," *Japanese Scientific Monthly*, vol. 58, no. 2, pp. 48–54, 2005.

[12] H. Hauser, A. J. Ijspeert, R. M. Füchslin, R. Pfeifer, and W. Maass, "Towards a theoretical foundation for morphological computation with compliant bodies," *Biological cybernetics*, vol. 105, no. 5-6, pp. 355–370, 2011.

[13] D. Verstraeten, B. Schrauwen, M. d'Haene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Networks*, vol. 20, no. 3, pp. 391–403, 2007.

[14] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, p. 34, 2001.

[15] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[16] C. Fernando and S. Sojakka, "Pattern recognition in a bucket," in *Advances in artificial life*. Springer, 2003, pp. 588–597.

[17] K. Vandoorne, P. Mechet, T. Van Vaerenbergh, M. Fiers, G. Morthier, D. Verstraeten, B. Schrauwen, J. Dambre, and P. Bienstman, "Experimental demonstration of reservoir computing on a silicon photonics chip," *Nature communications*, vol. 5, 2014.

[18] K. Caluwaerts, M. D'Haene, D. Verstraeten, and B. Schrauwen, "Locomotion without a brain: physical reservoir computing in tensegrity structures," *Artificial life*, vol. 19, no. 1, pp. 35–66, 2013.

[19] J. Degrave, R. Van Cauwenbergh, F. wyffels, T. Waegeman, and B. Schrauwen, "Terrain classification for a quadruped robot," in *International Conference on Machine Learning and Applications*, 2013.

[20] Q. Zhao, K. Nakajima, H. Sumioka, H. Hauser, and R. Pfeifer, "Spine dynamics as a computational resource in spine-driven quadruped locomotion," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013, pp. 1445–1451.

[21] J. Dambre, D. Verstraeten, B. Schrauwen, and S. Massar, "Information processing capacity of dynamical systems," *Scientific Reports*, vol. 2, 2012.

[22] J. M. Cioffi and T. Kailath, "Fast, recursive-least-squares transversal filters for adaptive filtering," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 32, no. 2, pp. 304–337, 1984.

[23] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[24] J. Degrave, M. Burm, T. Waegeman, F. wyffels, and B. Schrauwen, "Comparing trotting and turning strategies on the quadrupedal oncilla robot," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2013.

[25] J. Degrave, M. Burm, P.-J. Kindermans, J. Dambre, and F. wyffels, "Transfer learning of gaits on a quadrupedal robot," *Adaptive Behavior*, Prepublished January 20 2015.

7

A Differentiable Physics Engine for Deep Learning in Robotics

J. Degrave, M. Hermans, J. Dambre, and F. wyffels, "A differentiable physics engine for deep learning in robotics," *arXiv preprint arXiv:1611.01652*, 2016

⋆⋆⋆

In this chapter, we take a look at the culmination of combining robot morphology and controller into one system to optimize. Currently, robots are often treated as a black box in this optimization process, which is the reason why derivative-free optimization methods such as evolutionary algorithms or reinforcement learning are omnipresent. When gradient-based methods are used, models are kept small or rely on finite difference approximations for the Jacobian. This method quickly grows expensive with increasing numbers of parameters, such as found in deep learning.

We propose an implementation of a modern rigid body physics engine, which can differentiate control parameters. This engine is implemented for both CPU and GPU. Using this engine, we can backpropagate through time in our controller, through the physics of our engine and also through the rendering process. This chapter shows how such an engine speeds up the optimization process, even for small problems. We argue that this is a big step for deep learning in robotics, as it opens up new possibilities to optimize robots, both in hardware and software. By backpropagating the error through a model of the robot, we are essentially transferring prior knowledge on the model of the robot into the neural network controller. We are able to show that it only takes a small number of update steps before the robot

can learn to control a simple setup through vision, despite that the controller has around one million parameters.

7.1 Introduction

To solve tasks efficiently, robots require an optimization of their control system. This optimization process can be done in automated testbeds [2], but typically these controllers are optimized in simulation. Standard methods to optimize these controllers include particle swarms, reinforcement learning, genetic algorithms and evolutionary strategies. These are all derivative-free methods.

A recently popular alternative approach is to use deep Q-learning, a reinforcement learning algorithm. This method requires a lot of evaluations in order to train the many parameters [3]. However, deep learning experience has taught us that optimizing with a gradient is often faster and more efficient. This fact is especially true when there are a lot of parameters, as is common in deep learning. However, in the optimization processes for control systems, the robot is almost exclusively treated as a non-differentiable black box. The reason for this is that the robot in hardware is not differentiable, nor are current physics engines able to provide the gradient of the robot models. The resulting need for derivative-free optimization approaches limits both the optimization speed and the number of parameters in the controllers.

Recent physics engines, such as mujoco [4], can derive gradients through the model of a robot but rely on a finite difference method to approximate the gradient. Evaluating finite difference approximations, however, requires the same number of model evaluations as the number of states with respect to which is differentiated. Additionally, the gradient is an estimation.

In this paper, we suggest an alternative approach, by introducing a differentiable physics engine with analytical gradients. This idea is not novel. It has been done before with spring-damper models in 2D and 3D [5]. This technique is also similar to adjoint optimization, a method widely used in various applications such as thermodynamics [6] and fluid dynamics [7]. However, modern engines to model

robotics are not based on spring-damper systems. The most commonly used ones are 3D rigid body engines, which rely on impulse-based velocity stepping methods [8]. In this paper, we test whether these engines are also differentiable and whether this gradient is computationally tractable. We will show how this method does speed up the optimization process tremendously, and give some examples where we optimize deep learned neural network controllers with millions of parameters.

7.2 A 3D Rigid Body Engine

The goal is to implement a modern 3D Rigid body engine, in which parameters can be differentiated with respect to the fitness a robot achieves in a simulation, such that these parameters can be optimized with methods based on gradient descent.

The most frequently used simulation tools for model-based robotics, such as PhysX, Bullet, Havok and ODE, go back to MathEngine [8]. These tools are all 3D rigid body engines, where bodies have 6 degrees of freedom, and the relations between them are defined as constraints. These bodies exert impulses on each other, but their positions are constrained, e.g. to prevent the bodies from penetrating each other. The velocities, positions and constraints of the rigid bodies define a linear complementarity problem (LCP) [9], which is then solved using a Gauss-Seidel projection (GSP) method [10]. The solution of this problem are the new velocities of the bodies, which are then integrated by semi-implicit Euler integration to get the new positions [11]. This system is not always numerically stable. Therefore the constraints are usually softened [12].

The recent development of general automatic differentiation libraries such as Theano [13], Caffe [14] and Tensorflow [15], has allowed for efficient differentiation of remarkably complex functions before [16]. Therefore, we implemented such a physics engine as a mathematical expression in Theano, a software library which does automatic evaluation and differentiation of expressions with a focus on deep learning. The resulting computational graph to evaluate this expression is then compiled for both CPU and GPU. To be able to compile for GPU however, we had to limit our implementation to a restricted

set of elementary operations. The range of implementable functions is therefore severely capped. However, since the analytic gradient is determined automatically, the complexity of correctly implementing the differentiation is removed entirely.

One of these limitations with this restricted set of operations, is the limited support for conditionals. Therefore we needed to implement our physics engine without branching, as this is not yet available in Theano for GPU. Therefore some sacrifices had to be made. For instance, our system only allows for contact constraints between different spheres or between spheres and the ground plane. Collision detection algorithms for cubes typically have a lot of branching [17]. However, this sphere based approach can in principle be extended to any other shape [18]. On the other hand, we did implement a rather accurate model of servo motors, with gain, maximal torque, and maximal velocity parameters.

Another design choice was to use rotation matrices rather than the more common quaternions for representing rotations. Consequently, the states of the bodies are larger, but the operations required are matrix multiplications. This design reduced the complexity of the graph. However, cumulative operations on a rotation matrix might move the rotation matrix away from orthogonality. To correct for this, we renormalize our matrix with the update equation [19]:

$$A' = \frac{3A - A \circ (A \cdot A)}{2} \tag{7.1}$$

where A' is the renormalized version of the rotation matrix A. '$\circ$' denotes the elementwise multiplication, and '$\cdot$' the matrix multiplication.

These design decisions are the most important aspects of difference with the frequently used simulation tools. In the following section, we will evaluate our physics simulator on some different problems. We take a look at the speed of computation and the number of evaluations required before the parameters of are optimized.

7.2.1 Throwing a Ball

To test our engine, we implemented the model of a giant soccer ball in the physics engine, as shown in Fig. 7.3a. The ball has a 1 m diameter, a friction of $\mu = 1.0$ and restitution $e = 0.5$. The ball starts off at position $(0, 0)$. After 5 s it should be at position $(10, 0)$ with zero velocity v and zero angular velocity ω. We optimized the initial velocity v_0 and angular velocity ω_0 at time $t = 0$ s until the errors at $t = 5$ s are less than 0.01 m and 0.01 m/s respectively.

Since the quantity we optimize is only know at the end of the simulation, but we need to optimize the parameters at the beginning of the simulation, we need to backpropagate our error through time (BPTT) [20]. This approach is similar to the backpropagation through time method used for optimizing recurrent neural networks (RNN). In our case, every time step in the simulation can be seen as one pass through a neural network, which transforms the inputs from this timestep to inputs for the next time step. For finding the gradient, this RNN is unfolded completely, and the gradient can be obtained by differentiating this unfolded structure. This analytic differentiation is done automatically by the Theano library.

Optimizing the six parameters in v_0 and ω_0 took only 88 iterations with gradient descent and backpropagation through time. Optimizing this problem with CMA-ES [21], a state of the art derivative-free optimization method, took 2422 iterations. Even when taking the time to compute the gradient into account, the optimization with gradient descent takes 16.3 s, compared to 59.9 s with CMA-ES. This result shows that gradient-based optimization of kinematic systems can in some cases already outperform gradient-free optimization algorithms from as little as six parameters.

7.3 Policy Search

To evaluate the relevance of our differentiable physics engine, we use a neural network as a general controller for a robot, as shown in Figure 7.1. We consider a general robot model in a discrete-time dynamical system $\mathbf{x}^{t+1} = f_{\text{ph}}(\mathbf{x}^t, \mathbf{u}^t)$ with a task cost function of

$l(\mathbf{x}^t, \mathbf{p})$, where $\mathbf{x}^t$ is the state of the system at time t and $\mathbf{u}^t$ is the input of the system at time t. $\mathbf{p}$ provides some freedom in parameterizing the loss. If X^t is the trajectory of the state up to time $t-1$, the goal is to find a policy $u^t = \pi(X^t)$ such that we minimize the loss $\mathcal{L}_\pi$.

$$\begin{aligned} \mathcal{L}_\pi &= \sum_{t=0}^{T} l(\mathbf{x}^t, \mathbf{p}) \\ \text{s.t.} \quad \mathbf{x}^{t+1} &= f_{\text{ph}}(\mathbf{x}^t, \pi(X^t)) \quad \text{and} \quad \mathbf{x}^0 = x^{\text{init}} \end{aligned} \tag{7.2}$$

In previous research, finding a gradient for this objective has been described as presenting challenges [22]. An approximation to tackle these issues has been discussed in [23].

We implement this equation into an automatic differentiation library, ignoring these challenges in finding the analytic gradient altogether. The automatic differentiation library, Theano in our case, analytically derives this equation and compiles code to evaluate both the equation and its gradient. It does this by taking the computational graph of the equation, and constructing the computational graph of the requested gradient with the chain rule. Consequently, it compiles this graph into machine code in order to evaluate both efficiently, for both CPU and GPU.

We define our controller as a deep neural network g_{deep} with weights $\mathbf{W}$. We do not pass all information X^t to this neural network, but only a vector of values $\mathbf{s}^t$ observed by the modeled sensors $s(\mathbf{x}^t)$. We also provide our network with (some of the) task-specific parameters $\mathbf{p}'$. Finally, we add a recurrent connection to the controller in the previous timestep $\mathbf{h}^t$. Therefore, our policy is the following:

$$\begin{aligned} \pi(X^t) &= g_{\text{deep}}(s(\mathbf{x}^t), \mathbf{h}^t, \mathbf{p}' \mid \mathbf{W}) \\ \text{s.t.} \quad \mathbf{h}^t &= h_{\text{deep}}(s(\mathbf{x}^{t-1}), \mathbf{h}^{t-1}, \mathbf{p}' \mid \mathbf{W}) \quad \text{and} \quad \mathbf{h}^0 = 0 \end{aligned} \tag{7.3}$$

Notice the similarity between equations 7.2 and 7.3. Indeed, the equations for recurrent neural networks (RNN) in equation 7.3 are very similar to the ones of the loss of a physical model in equation 7.2. Therefore, we optimize this entire system as an RNN unfolded over time, as illustrated in Figure 7.2. The weights $\mathbf{W}$ are optimized with stochastic gradient descent. The gradient required for that is the Jaco-

bian $d\mathcal{L}/d\mathbf{W}$, which is found with automatic differentiation software.

We have now reduced the problem to a standard deep learning problem. We need to train our network g_{deep} on a sufficient amount of samples x^{init} and for a sufficient amount of sampled tasks $\mathbf{p}$ in order to get adequate generalization. Standard RNN regularization approaches could also improve this generalization. We reckon that generalization of g_{deep} to more models f_{ph}, in order to ease the transfer of the controller from the model to the real system, is also possible [5], but it is outside the scope of this paper.

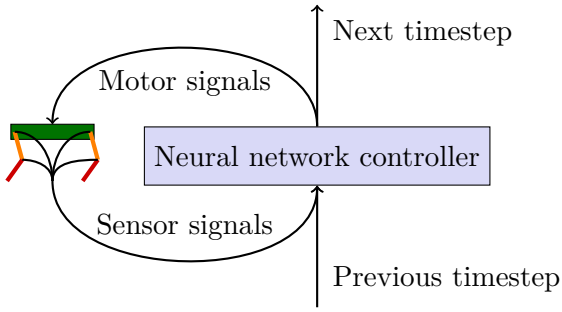


Fig. 7.1: Illustration of how a closed loop neural network controller would be used to actuate a robot. The neural network receives sensor signals from the sensors on the robot and uses these to generate motor signals which are sent to the servo motors. The neural network can also generate a signal which it can use at the next timestep to control the robot.

7.3.1 Quadrupedal Robot – Computing Speed

To verify the speed of our engine, we also implemented a small quadrupedal robot model, as illustrated in Fig. 7.3b. This model has a total of 81 sensors, e.g. encoders and an inertial measurement unit (IMU). The servo motors are controlled in a closed loop by a small neural network g_{deep} with a number of parameters, as shown previously in Fig. 7.2. The gradient is the Jacobian of $\mathcal{L}$, the total traveled distance of the robot in 10 s, differentiated with respect to all the parameters of the controller $\mathbf{W}$. This Jacobian is found by using BPTT and propagating all 10 s back. The time it takes to compute this traveled distance and the accompanying Jacobian is shown

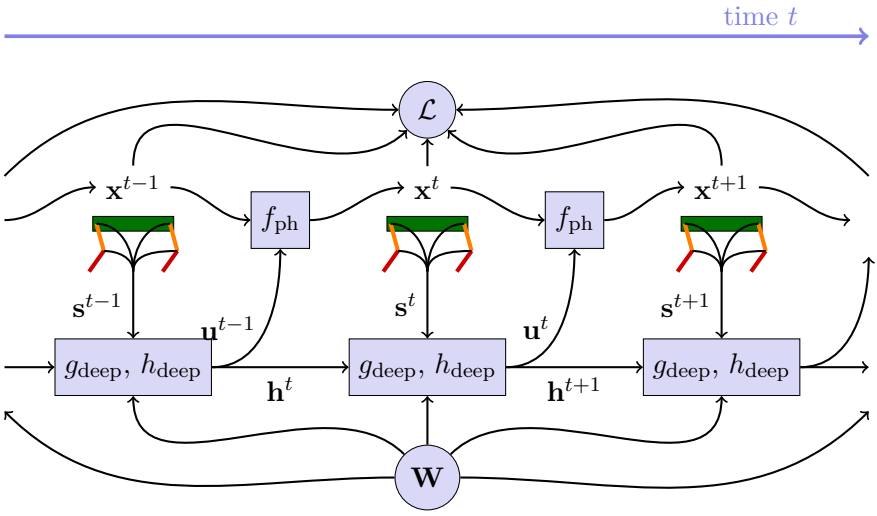
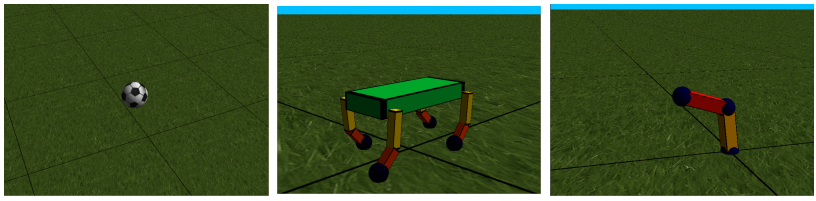


Fig. 7.2: Illustration of the dynamic system with the robot and controller, after unrolling over time. The neural networks g_{deep} and h_{deep} with weights $\mathbf{W}$ receive sensor signals $\mathbf{s}^t$ from the sensors on the robot and use these to generate motor signals $\mathbf{u}^t$ which are used by the physics engine f_{ph} to find the next state of the robot in the physical system. These neural networks also have a memory, implemented with recurrent connections $\mathbf{h}^t$. From the state $\mathbf{x}^t$ of these robots, the loss $\mathcal{L}$ can be found. In order to find $d\mathcal{L}/d\mathbf{W}$, every block in this chart needs to be differentiable. The contribution of this paper, is to implement a differentiable f_{ph}, which allows us to optimize $\mathbf{W}$ to minimize $\mathcal{L}$ more efficiently than was possible before.

in Table 7.1. We include both the computation time with and without the gradient, i.e. both the forward and backward pass and the forward pass alone. This way, the numbers can be compared to other physics engines, as those only calculate without gradient. Our implementation and our model can probably be made more efficient, and evaluating the gradient can probably be made faster a similar factor.



(a) Ball model **(b)** quadruped model **(c)** robot arm model

Fig. 7.3: (a) Illustration of the ball model used in the first task. (b) Illustration of the quadruped robot model with 8 actuated degrees of freedom, 1 in each shoulder, 1 in each elbow. The spine of the robot can collide with the ground, through 4 spheres in the inside of the cuboid. (c) Illustration of the robot arm model with 4 actuated degrees of freedom.

When only a single controller is optimized, our engine runs more slowly on GPU than on CPU. To tackle this issue, we implemented batch gradient descent, which is commonly used in complex optimization problems. In this case, by batching our robot models, we achieve significant acceleration on GPU. Although backpropagating the gradient through physics slows down the computations by roughly a factor 10, this factor only barely increases with the number of parameters in our controller.

Combining this with our previous observation that fewer iterations are needed when using gradient descent, our approach can enable the use of gradient descent through physics for highly complex deep neural network controllers with millions of parameters. Also note that by using a batch method, a single GPU can simulate about 864 000 model seconds per day, or 86 400 000 model states. This should be plenty for deep learning. It also means that a single simulation step of a single robot, which includes collision detection, solving the LCP problem, integrating the velocities and backpropagating the gradient through it all, takes about 1 ms on average. Without the backpropagation,

this process is only about seven times faster.

7.3.2 4 Degree of Freedom Robot Arm

As a first test of optimizing robot controllers, we implemented a four degree of freedom robotic arm, as depicted in Fig. 7.3c. The bottom of the robot has a 2 degrees of freedom actuated universal joint; the elbow has a 2 degree of freedom actuated joint as well. The arm is 1 m long, and has a total mass of 32 kg. The servos have a gain of $30\,\mathrm{s}^{-1}$, a torque of 30 Nm and a velocity of $45^\circ\,\mathrm{s}^{-1}$.

For this robot arm, we train controllers for a task with a gradually increasing amount of difficulty. To be able to train our parameters, we have to use a couple of tricks often used in the training of recurrent neural networks.

- We choose an objective which is evaluated at every time step and then averaged, rather than at specific points of the simulation. This approach vastly increases the number of samples over which the gradient is averaged, which in turn makes the gradient direction more reliable [24].
- The value of the gradient is decreased by a factor $\alpha < 1$ at every time step. This trick has the effect of a prior. Namely, events further in the past are less important for influencing current events, because intermediate events might diminish their influence altogether. It also improves robustness against exploding gradients [5].
- We initialize the controller intelligently. We do not want the controller to shake the actuators violently and explore outside the accurate domain of our simulation model. Therefore our controllers are initialized such that they only output zeros at the start of the simulation. The initial policy is the zero policy.
- We constraint the size of the gradient to an L2-norm of 1. This makes sure that gradients close to discontinuities in the fitness landscape do not push the parameter values too far away, such that everything which was learned is forgotten [20].

Tab. 7.1: Evaluation of the computing speed of our engine on a robot model controlled by a closed loop controller with a variable number of parameters. We evaluated both on CPU (i7 5930K) and GPU (GTX 1080), both for a single robot optimization and for batches of multiple robots in parallel. The numbers are the time required in seconds for simulating the quadruped robot(s) for $10\,\mathrm{s}$, with and without updating the controller parameters through gradient descent. The gradient calculated here is the Jacobian of the total traveled distance of the robot in $10\,\mathrm{s}$, differentiated with respect to all the parameters of the controller. For comparison, the model has 102 states. It is built from 17 rigid bodies, each having 6 degrees of freedom. These states are constrained by exactly 100 constraints.

Seconds of computing time required to simulate a batch of robots for 10 seconds

#robots	#parameters	with gradient		without gradient	
		CPU	GPU	CPU	GPU
1	1 296	8.17	69.6	1.06	9.69
	1 147 904	13.2	75.0	2.04	9.69
128	1 296	263	128	47.7	17.8
	1 147 904	311	129	50.4	18.3

Milliseconds of computing time required to perform one time step of one robot.

#robots	#parameters	with gradient		without gradient	
		CPU	GPU	CPU	GPU
1	1 296	8.17	69.6	1.06	9.69
	1 147 904	13.2	75.0	2.04	9.69
128	1 296	2.05	1.00	0.372	0.139
	1 147 904	2.43	1.01	0.394	0.143

7.3.2.1 Reaching a Fixed Point

A first simple task, is to have a small neural net controller learn to move the controller to a certain fixed point in space, at coordinates $(0.5\,\mathrm{m}; 0.5\,\mathrm{m}; 0.5\,\mathrm{m})$. The objective we minimize for this task, is the distance between the end effector and the target point, averaged over the 8 seconds we simulate our model.

We provide the controller with a single sensor input, namely the current distance between the end effector and the target point. Input is not required for this task, as there are solutions for which the motor signals are constant in time. However, this would not necessarily be the optimal approach for minimizing the average distance over time, it only solves the distance at the end of the simulation, but does not minimize the distance during the trajectory to get at the final position.

As a controller, we use a dense neural network with 1 input, 2 hidden layers of 128 units with a rectifier activation function, and 4 outputs with an identity activation function. This controller has 17 284 parameters in total. We disabled the recurrent connections $\mathbf{h}^t$.

We use gradient descent with a batch size of 1 robot for optimization, as the problem is not stochastic in nature. The parameters are optimized with Adam's rule [25] with a learning rate of 0.001. Every update step with this method takes about 5 seconds on CPU. We find that the controller comes within 4 cm of the target in 100 model evaluations, and within 1 cm in 150 model evaluations, which is small compared to the 1 m arm of the robot. Moreover, the controller does find a more optimal trajectory which takes into account the sensor information.

Solving problems like these in fewer iteration steps than the number of parameters, is unfeasible with derivative free methods [24]. Despite that, we did try to optimize the same problem with CMA-ES. After a week of computing and 60 000 model evaluations, CMA-ES did not show any sign of convergence, as it cannot handle the sheer amount of parameters.

7.3.2.2 Reaching a Random Point

As a second task, we sample a random target point in the reachable space of the end effector. We give this point as input v' to the controller, and the task is to again minimize the average distance between the end effector and the target point v. Our objective $\mathcal{L}$ is this distance averaged over all timesteps.

As a controller, we use a dense neural network comparable to the previous section, but this time with 3 inputs. We used 3 hidden layers with 1024 units each, so the controller has 2 107 396 parameters in total. This is not necessary for this task, but we do it like this to demonstrate the power of this approach. In order to train for this task, we use a batch size of 128 robots, such that every update step takes 58 s on GPU. Each simulation takes 8 s with a simulation step of 0.01 s. Therefore, the gradient on the parameters of the controllers has been averaged over 51 200 timesteps at every update step. We update the parameters with Adam's rule, where we scale the learning rate with the average error achieved in the previous step.

We find that it takes 576 update steps before the millions of parameters are optimized, such that the end effector of the robot is on average less than 10 cm of target, 2 563 update steps before the error is less than 5 cm.

7.3.3 A Quadrupedal Robot – Revisited

Optimizing a gait for a quadrupedal robot is a problem of a different order, something the authors have extensive experience with [26, 27, 2]. The problem is way more challenging and allows for a broad range of possible solutions. In nature, we find a wide variety of gaits, from hopping over trotting, walking and galloping. With hand tuning on the robot model shown in Figure 7.3b, we were able to obtain a trotting motion with an average forward speed of 0.7 m/s. We found it tricky to find a gait where the robot did not end up like an upside down turtle, as 75% of the mass of the robot is located in its torso.

As a controller for our quadrupedal robot, we use a neural network with 2 input signals $\mathbf{s}^t$, namely a sine and a cosine signal with a

frequency of 1.5 Hz. On top of this, we added 2 hidden layers of 128 units and a rectifier activation function. As output layer, we have a dense layer with 8 units and a linear activation function, which has as input both the input layer and the top layer of the hidden layers. In total, this controller has 17 952 parameters. Since the problem is not stochastic in nature, we use a batch size of 1 robot. We initialize the output layer with zero weights, so the robot starts the optimization in a stand still position.

We optimize these parameters to maximize the average velocity of the spine over the course of 10 s of time in simulation. This way, the gradient used in the update step is effectively an average of the 1 000 time steps after unrolling the recurrent connections. This objective does not take into account energy use, or other metrics typically employed in robotic problems.

In only 500 model evaluations or about 1 hour of optimizing on CPU, the optimization with BPTT comes up with a solution with a speed of 1.17 m/s. This solution is a hopping gait, with a summersault every 3 steps[1], despite limiting the torque of the servos to 4 Nm on this 28.7 kg robot. For more life-like gaits, energy efficiency could be use as a regularization method. Evaluating these improvements are however outside the scope of this paper.

7.3.3.1 The inverted pendulum with a camera as sensor

As a fourth example, we implemented a model of the pendulum-cart system we have in our laboratorium. This pendulum-cart system is used for the classic control task of the underactuated inverted pendulum [28]. In this example however, a camera which is set up in front of the system is the only available information for the controller. It therefore has to observe the system it controls using vision. A frame captured by this camera is shown in Figure 7.4.

In order to build this model, we implemented a differentiable camera in our physics engine. This camera uses a ray-tracing approach to find where it needs to sample from the textures, and uses bilinear interpolation to sample from these textures, similar to the one used

[1] A video is available at `https://goo.gl/5ykZZe`

for the spatial transform layer [29]. This interpolation is necessary for making the frame captured by the camera differentiable to the state of the robot with non-zero derivatives.

Fig. 7.4: A frame captured by the differentiable camera looking at the model of the pendulum-cart system. The resolution used is 288 by 96 pixels. All the textures are made from pictures of the actual system.

We minimize the distance from the end of the pendulum to the desired point and regularize the speed of the pendulum. The memoryless deep controller receives the current image of the camera, in addition to two images from the past such that it can estimate velocity and acceleration. We observe that a controller with 1,065,888 parameters is able to learn to swing up and keep the pendulum stable from pixels after only 2420 episodes of 3 model seconds. The complete optimization process took 15 hours. The resulting controller keeps the pendulum stable for more than one minute[2]. In order to do this, the controller has learned to interpret the frames it receives from the camera and found a suitable control strategy.

7.4 Discussion

Our results show the first prototype of a differentiable physics engine based on similar algorithms as those that are commonly used in modern robotics simulators. When initially addressing the problem, we did not know whether finding the gradient would be computationally tractable, let alone whether evaluating it would be fast enough to be

[2]https://twitter.com/317070/status/821062814798331905

beneficial for optimization. In this paper, we have demonstrated that evaluating the gradient is tractable enough to speed up optimization on problems with as little as six parameters. The speed of this evaluation mainly depends on the complexity of the physics model and only slightly on the number of parameters to optimize. Therefore, our results suggest that this cost is dominated by the gain achieved from the combination of using batch gradient descent and GPU acceleration.

Optimizing the controller of a robot model with gradient-based optimization is equivalent to optimizing an RNN. After all, the gradient passes through each parameter at every time step. The parameter space is therefore very noisy. Consequently, training the parameters of this controller is a highly non-trivial problem, as it corresponds to training the parameters of an RNN. On top of that, exploding and vanishing signals and gradients cause far more challenging problems compared to feed forward networks.

In section 7.3.2, we already discussed some of the tricks used for optimizing RNNs. Earlier research shows that these methods can be extended to more complicated tasks than the ones discussed here [5, 20]. Hence, we believe that this approach towards learning controllers for robotics applies to more complex problems than the illustrative examples in this paper.

All of the results in this paper will largely depend on showing how these controllers will work on the physical counterparts of our models. Nonetheless, we would like to conjecture that to a certain extent, this gradient of a model is close to the gradient of the physical system. The gradient of the model is more susceptible to high-frequency noise introduced by modeling the system, than the imaginary gradient of the system itself. Nonetheless, it contains information which might be indicative, even if it is not perfect. We would theorize that using this noisy gradient is still better than optimizing blindly and that the transferability to real robots can be improved by evaluating the gradients on batches of (slightly) different robots in (slightly) different situations and averaging the results. This technique has already been applied in [5] as a regularization method to avoid bifurcations during online learning. If the previous proves to be correct, our approach can offer an addition or possibly even an alternative to deep Q-learning for deep neural network controllers in robotics.

We can see the use of this extended approach for a broad range of applications in robotics. Not only do we think there are multiple ways where recent advances in deep learning could be applied to robotics more efficiently with a differentiable physics engine, we also see various ways in which this engine could improve existing angles at which robotics are currently approached:

- In this paper, we added memory by introducing recurrent connections in the neural network controller. We reckon that advanced, recurrent connections such as ones with a memory made out of LSTM cells [30] can allow for more powerful controllers than the controllers described in this paper.
- Using a differentiable physics engine, we reckon that knowledge of a model can be transferred more efficiently into a forward or backward model in the form of a neural network, similar to methods such as used in [31] and [32]. By differentiating through an exact model and defining a relevant error on this model, it should be possible to transfer knowledge from a forward or backward model in the differentiable physics engine to a forward or backward neural network model. Neural network models trained this way might be more robust than the ones learned from generated trajectories [33]. In turn, this neural model could then be used for faster but approximate evaluation of the model.
- Although we did not address this in this paper, there is no reason why only control parameters could be differentiated. Hardware parameters of the robot have been optimized the same way before [6, 7, 5]. The authors reckon that the reverse process is also true. A physics engine can provide a strong prior, which can be used for robots to learn (or adjust) their robot models based on their hardware measurements faster than today. You could optimize the model parameters with gradient descent through physics, to have the model better mimic the actual observations.
- Where adversarial networks are already showing their use in generating image models, we believe adversarial robotics training (ART) will create some inventive ways to design and control robots. Like in generative adversarial nets (GAN) [34], where the gradient is pulled through two competing neural

networks, the gradient could be pulled through multiple competing robots as well. It would form an interesting approach for swarm robotics, similar to previous results in evolutionary robotics [35, 36, 37], but possibly faster.

7.5 Conclusion

In this paper, we show it is possible to build a differentiable physics engine. We implemented a modern engine which can run a 3D rigid body model, using the same algorithm as other engines commonly used to simulate robots, but we can additionally differentiate control parameters with BPTT. Our implementation also runs on GPU, and we show that using GPUs to simulate the physics can speed up the process for large batches of robots. We show that even complex sensors such as cameras, can be implemented and differentiated through, allowing for computer vision to be learned together with a control policy.

We find that these gradients can be computed fast enough for use in applications. We also show that using gradient descent with BPTT speeds up optimization processes often found in robotics, even for rather small problems, due to the reduced number of model evaluations required. We show that this improvement in speed scales to problems with a lot of parameters. We also show that using this engine, finding policies for robot models can be done faster and in a more straightforward way. This method should allow for a new approach to apply deep learning techniques in robotics.

Acknowledgments

Special thanks to David Pfau for pointing out relevant prior art we were previously unaware of, and Iryna Korshunova for proofreading the paper. The research leading to these results has received funding from the Agency for Innovation by Science and Technology in Flanders (IWT). The NVIDIA Corporation donated the GTX 1080 used for this research.

Bibliography

[1] J. Degrave, M. Hermans, J. Dambre, and F. wyffels, "A differentiable physics engine for deep learning in robotics," *arXiv preprint arXiv:1611.01652*, 2016.

[2] J. Degrave, M. Burm, P.-J. Kindermans, J. Dambre, and F. wyffels, "Transfer learning of gaits on a quadrupedal robot," *Adaptive Behavior*, p. 1059712314563620, 2015.

[3] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *arXiv preprint arXiv:1603.02199*, 2016.

[4] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[5] M. Hermans, B. Schrauwen, P. Bienstman, and J. Dambre, "Automated design of complex dynamic systems," *PloS one*, vol. 9, no. 1, p. e86696, 2014.

[6] Y. Jarny, M. Ozisik, and J. Bardon, "A general optimization method using adjoint equation for solving multidimensional inverse heat conduction," *International journal of heat and mass transfer*, vol. 34, no. 11, pp. 2911–2919, 1991.

[7] A. Iollo, M. Ferlauto, and L. Zannetti, "An aerodynamic optimization method based on the inverse problem adjoint equations," *Journal of Computational Physics*, vol. 173, no. 1, pp. 87–115, 2001.

[8] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of Bullet, Havok, Mujoco, ODE and PhysX," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 4397–4404.

[9] D. Chappuis, "Constraints derivation for rigid body simulation in 3D," 2013.

[10] F. Jourdan, P. Alart, and M. Jean, "A Gauss-Seidel like algorithm to solve frictional contact problems," *Computer methods in applied mechanics and engineering*, vol. 155, no. 1, pp. 31–47, 1998.

[11] D. Stewart and J. C. Trinkle, "An implicit time-stepping scheme for rigid body dynamics with coulomb friction," in *International Conference on Robotics and Automation (ICRA)*, vol. 1. IEEE, 2000, pp. 162–169.

[12] E. Catto, "Modeling and solving constraints," in *Game Developers Conference*, 2009.

[13] R. Al-Rfou, G. Alain, *et al.*, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[15] M. Abadi, A. Agarwal, *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015.

[16] J. Degrave, S. Dieleman, J. Dambre, and F. wyffels, "Spatial chirp-Z transformer networks," in *European Symposium on Artificial Neural Networks (ESANN)*, 2016.

[17] B. Mirtich, "V-clip: Fast and robust polyhedral collision detection," *ACM Transactions On Graphics (TOG)*, vol. 17, no. 3, pp. 177–208, 1998.

[18] P. M. Hubbard, "Approximating polyhedra with spheres for time-critical collision detection," *ACM Transactions on Graphics (TOG)*, vol. 15, no. 3, pp. 179–210, 1996.

[19] W. Premerlani and P. Bizard, "Direction cosine matrix IMU: Theory," *DIY Drone: USA*, pp. 13–15, 2009.

[20] I. Sutskever, "Training recurrent neural networks," Ph.D. dissertation, University of Toronto, 2013.

[21] N. Hansen, "The CMA evolution strategy: a comparing review," in *Towards a new evolutionary computation.* Springer Berlin Heidelberg, 2006, pp. 75–102.

[22] I. Mordatch and E. Todorov, "Combining the benefits of function approximation and trajectory optimization," in *Robotics: Science and Systems (RSS)*, 2014.

[23] S. Levine and V. Koltun, "Variational policy search via trajectory optimization," in *Advances in Neural Information Processing Systems*, 2013, pp. 207–215.

[24] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky, "Nonlinear black-box modeling in system identification: a unified overview," *Automatica*, vol. 31, no. 12, pp. 1691–1724, 1995.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

[26] A. Sproewitz, A. Tuleu, M. D'Haene, R. Möckel, J. Degrave, M. Vespignani, S. Gay, M. Ajallooeian, B. Schrauwen, and A. J. Ijspeert, "Towards dynamically running quadruped robots: performance, scaling, and comparison," in *Adaptive Motion of Animals and Machines*, 2013, pp. 133–135.

[27] J. Degrave, M. Burm, T. Waegeman, F. Wyffels, and B. Schrauwen, "Comparing trotting and turning strategies on the quadrupedal oncilla robot," in *IEEE International Conference on Robotics and Biomimetics (ROBIO).* IEEE, 2013, pp. 228–233.

[28] R. J. Vaccaro, *Digital control: a state-space approach.* McGraw-Hill New York, 1995, vol. 196.

[29] M. Jaderberg, K. Simonyan, A. Zisserman *et al.*, "Spatial transformer networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2017–2025.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[31] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," *arXiv preprint arXiv:1603.08155*, 2016.

[32] V. Dumoulin, J. Shlens, and M. Kudlur, "A learned representation for artistic style," *CoRR*, vol. abs/1610.07629, 2016.

[33] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.

[34] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[35] K. Sims, "Evolving 3D morphology and behavior by competition," *Artificial life*, vol. 1, no. 4, pp. 353–372, 1994.

[36] R. Pfeifer and J. Bongard, *How the body shapes the way we think: a new view of intelligence.* MIT press, 2006.

[37] N. Cheney, J. Clune, and H. Lipson, "Evolved electrophysiological soft robots," in *ALIFE*, vol. 14, 2014, pp. 222–229.

8 Conclusions & Perspectives

In this chapter, we look back at the main observations made during my research, such that the overarching trajectory of my research emerges. Furthermore, by extrapolating this trajectory, the section on perspectives will provide insight as to what I think are interesting directions for future exploration.

8.1 Conclusions

The overarching focus of this dissertation is the study of adding prior information to neural networks, especially in the context of robotics. We have discussed various approaches and gained insight into the principles underlying these methods. For each of these situations, we had delved into why they work – or more interestingly, when and why they did not – and assessed their use for future approaches when controlling robots with neural networks.

In chapter 2, we discussed a way to encode prior knowledge into the architecture of neural networks, in this case for the classification of images. We start off with a known approach to learning invariances for images, i.e. image augmentation. We build upon this idea to set up equations where you can backpropagate through the augmentation process instead. This way, knowledge which before had to be encoded into the parameters of the network, could now be encoded in the architecture. We showed that this approach indeed outperforms standard convolutional neural networks.

In chapter 3, we turned our focus to robotics and laid down the groundwork for the later chapters. We introduced the robot *Oncilla* and developed three different gaits. Based on those, we uncovered a few principles for legged robot control. Firstly, we observed that a more biologically inspired trajectory outperforms a sine-based approach, especially for higher speeds. Secondly, we showed that having scapulae for turning helps maintain speed, although it is also possible for legged robots to turn without. Thirdly, we showed that it is feasible to optimize a legged robot without building a model when enough prior information is available for the optimization process such that only a limited number of parameters need tuning.

In chapter 4, we continued on these observations and evaluated transfer learning as an approach for including prior knowledge in the learning process of gaits on legged robots. In order to test our hypothesis, we ran a gait optimization process on flat terrain. Then, we ran multiple optimization procedures in various setups varying morphological parameters and the environment. To transfer the parameters obtained on the flat terrain to these new setups, we use warm starts from the parameters and evaluate the optimization process against what we would obtain when not having this warm start. Everything considered, we conclude that this transfer of knowledge did result in better gaits. Although an important footnote, in one of the cases the warm start did not make a significant difference, namely when transfer learning from a flat terrain to a rocky one.

We found that the main aspect which made these warm starts perform better was the reduced amount of exploration in the parameter space. This allowed the robot to quicker converge on an interesting part of the parameter space and consequently find better solutions in the limited amount of wall clock time available.

In chapter 5, we focused on using the morphology of the robot as a resource for sensing and computation. We showed that a limited but appropriate selection of input sensors on a legged robot is sufficient to perform terrain classification. However, we also showed the importance of non-linearities for achieving those results, and that adding recursion in the form of reservoir computing further improves the results. More interestingly, from the comparison of extreme learning machines to reservoirs, it seems clear that memory is an important element.

To follow up on those finding, in chapter 6 we dug deeper in this trade-off between memory and non-linearities. We developed an embodied control system and showed that this can generate a dynamically balanced trot on a compliant robot. For this, we trained a small neural network with regularized recurrent least squares to mimic the gaits we developed earlier. We demonstrated that in as little as a couple of periods of the gait, this system already finds stable feedback control parameters for trotting without requiring any memory. However, the amount of memory seems to trade-off with the amount of non-linearity required. We found that the most important factor is the number of signals fed to the linear transformation. We concluded that this approach is a powerful tool for transferring knowledge from an open loop trajectory into a closed loop neural network. It shows in particular how important the use of priors can be, by reducing an engineered trajectory to a simple relation between sensor input and motor signals.

Finally, in chapter 7 we take the multiple threads discussed in this book into a single approach, stretching the idea of morphological computation to its furthest extent. In this chapter, we treat the controller and the system as one source for computation and optimize the combined system with gradient descent. Getting the exact gradient of a system can however only be done in simulation, which is why we developed a physics engine inside a library for automatic differentiation. By using backpropagation through time, through physics and through the renderer, we were able to find a deep neural network controller for the inverted pendulum from pixels. For a quadruped legged robot model, we were able to use deep neural networks to find a stable gait in only 500 model evaluations. In this case, incorporating the gradients available from the model in the optimization of the controller is a strong indication of the power of priors in training neural network controllers. Therefore, we concluded that learning control from complex sensors such as cameras is achievable by incorporating a model of the system in the network architecture.

8.2 Perspectives

Once you have a differentiable physics engines, **co-optimizing the controller with its morphology** in simulation seems the first logical

step. Since the automatic differentiation library does not care whether a parameter is inside the morphology of the robot or is a parameter in the controller, both can be optimized simultaneously using gradient descent.

This approach seems especially promising, as it would allow going deeper into some questions posed by the idea of embodiment. It would allow for a thorough study whether optimizing the morphology such that it allows for more morphological computation gains you anything over optimizing the morphology for controllability, or whether in the end, both approaches obtain similar conclusions regarding optimal morphological parameters. This would be not unlike the original research [1] which was the inspiration for the backpropagation through physics.

Overall, physics engines are often treated by roboticists as a black box, similar to the treatment of the hardware of the robot. This is a useful mindset when assessing whether an approach will work on a robot, before actually trying it on a robot. However, there is value to be gained by putting the physics engines to full use and let them be a more powerful tool to tackle more fundamental open questions in robotics. They are fundamentally different tools. Complex models in physics engines can provide information which is not available when running the experiment on a robot, such as analytic gradients, reversing the time direction or variational sensitivity analysis. In turn, these can be used to analyze the base premises underlying the approaches to robotics.

If we can have analytic gradients of complex models, there is probably insight to be gained by comparing those gradients to the gradient approximations found by for instance policy gradient methods. Such an evaluation could shed some light on for instance the value of using natural gradients [2] in complex setups.

Bringing the controllers trained in simulation to a real environment is a step which can be tackled in multiple ways with a differentiable physics engine, and there are multiple possibilities there. For instance, since our entire model and controller are differentiable, we could optimize the model to mimic the data from the real environment as closely as possible. Such a thing could be done variationally, defining a distribution over the parameters of the model. For every

episode in simulation, a new model could then be sampled on which the controller is optimized. Such an approach would partially circumvent the problem of the model not being able to exactly mimic the physical setup. Secondly, the noise would make the controller more robust. But also only as robust as need be according to the variational model of the system.

Robot swarms are another point which could prove to gain from the differentiable model. By backpropagating through physics, the controllers in these swarms would learn directly to manipulate and communicate with each other as they are receiving the gradients flowing through the physics. It would allow them to train by gaming each other, competing for resources, but with a powerful signal of which direction to optimize in.

Injecting prior information into other approaches for learning control is a second possible extension to the work presented in this dissertation. We did not touch on ideas to add prior knowledge when using reinforcement learning for robotics yet, despite it being a research subject of increasing interest to the scientific community. It seems that remarkable results are possible with this approach, despite containing limited or no prior knowledge of the problem at hand [3]. That raises the question which results these approaches would be able to obtain when using more information. Especially since normally there is at least a vague description of the setup available, which should allow some prior knowledge from control theory to the initialize the search. After all, the main conclusion of the work presented here is that using more prior knowledge helps when training a neural network.

Bibliography

[1] M. Hermans, B. Schrauwen, P. Bienstman, and J. Dambre, "Automated design of complex dynamic systems," *PloS one*, vol. 9, no. 1, p. 86696, 2014.

[2] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on.* IEEE, 2006, pp. 2219–2225.

[3] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on.* IEEE, 2017, pp. 3389–3396.